universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

„Entwicklung eines Programmes zur Konstruktion logischer Schaltungen für den Informatikunterricht.“

Verfasser

Markus Bauer

angestrebter akademischer Grad

Magister der Naturwissenschaft (Mag.rer.nat.)

Wien, 2012

Studienkennzahl lt. Studienblatt:	A 190 412 884
Studienrichtung lt. Studienblatt:	UF Physik und UF Informatik und Informatikmanagement
Betreuer:	Univ.-Prof. Dr. Wilfried Grossmann

Abstract

Im Rahmen der Diplomarbeit wurde eine Software erstellt, die im Informatikunterricht zum Thema Boolesche Logik eingesetzt werden kann. Die Zielgruppe für das Programm sind Schülerinnen und Schüler der Schulstufen, in denen Informatik unterrichtet wird, also hauptsächlich im Alter zwischen 14 und 18 Jahren. Die Software ermöglicht es Schülerinnen und Schülern, das Erstellen von Gatterschaltungen zu üben. Benutzer des Programms haben die Möglichkeit, Gatterschaltungen selbst zu zeichnen oder vom Programm nach einer eingegebenen Booleschen Funktion zeichnen zu lassen. Boolesche Funktionen können entweder per Hand in das Programm eingegeben oder vom Programm aus einer gezeichneten Gatterschaltung berechnet werden. Boolesche Funktionen sowie Gatterschaltungen können von der Software verglichen und für verschiedene Belegungen der Eingangsvariablen verifiziert werden. In Abgrenzung zu einem reinen Lernprogramm soll es diese Software explizit nicht ermöglichen, das Thema Boolesche Logik von Grund auf zu erlernen. Vielmehr soll es aufbauend auf ein Grundwissen über Boolesche Logik und Gatterschaltungen ergänzend im Unterricht eingesetzt werden. Dafür enthält das Programm eine Funktion, mit Hilfe derer Aufgaben erstellt werden können, die dann von den Schülerinnen und Schülern bearbeitet werden. Die Resultate werden ebenfalls im Programm dargestellt.

Die Umsetzung des Projekts erfolgte in der Programmiersprache Java. Damit ist das Programm grundsätzlich plattformunabhängig und kann zum Beispiel auch auf einem Linuxcomputer eingesetzt werden. Gemäß dem Einsatzort Schule wurde bei der Entwicklung des Programms vor allem auf Robustheit und Korrektheit sowie Benutzerfreundlichkeit und Zuverlässigkeit Wert gelegt.

Inhaltsverzeichnis

1 Einleitung 7

2 Pflichtenheft 8

2.1 Zielbestimmungen 8

2.1.1 Musskriterien 8

2.1.2 Wunschkriterien 9

2.1.3 Abgrenzungskriterien 9

2.2 Produkteinsatz 9

2.2.1 Anwendungsbereiche 9

2.2.2 Zielgruppen 9

2.2.3 Betriebsbedingungen 10

2.3 Produktumgebung 10

2.3.1 Software 10

2.3.2 Hardware 10

2.4 Produktfunktionen 10

2.5 Produktdaten 11

2.6 Produktleistungen 11

2.7 Benutzeroberfläche 11

2.8 Qualitätsbestimmungen 12

2.9 Entwicklungsumgebung 12

2.9.1 Software und Tools 12

2.9.2 Hardware 12

3. Anwendungsfälle 13

3.1 Anwendungsfalldiagramm 13

3.2 Anwendungsfallbeschreibungen 14

4. Programmstruktur 24

5. Systemvoraussetzungen und Lizenzbedingungen 27

6. Benutzerhandbuch 28

6.1 Programmstart 28

6.2 Übersicht 28

6.3 Grundfunktionen 29

6.3.1 Gatterschaltung zeichnen 29

6.3.2 Gatterschaltung löschen 31
6.3.3 Boolesche Funktion eingeben 31
6.3.4 Speichern 31
6.3.5 Laden 32
6.3.6 Bildexport 32
6.4 Logische Funktionen 32
6.4.1 Wertebelegung 32
6.4.2 Automatische Evaluation 32
6.4.3 Vergleichen 33
6.4.4 Gatterschaltung als Funktion darstellen 33
6.4.5 Funktion als Gatterschaltung darstellen 33
6.5.6 Zufällige Funktion oder Gatterschaltung generieren. 33
6.5.7 Erfüllbarkeit überprüfen 33
6.5 Aufgaben 33
6.5.1 Arten von Aufgaben 33
6.5.2 Aufgaben erzeugen 34
6.5.3 Aufgaben bearbeiten 34
6.5.4 Aufgaben durchführen 34
6.5.5 Resultate anzeigen 35
7 Einsatzmöglichkeiten im Unterricht 36
8 Entwicklertagebuch 41
Abbildungen 45
Literatur 46
Javadoc 47
Lebenslauf

1 Einleitung

In meinem späteren Beruf als Informatiklehrer wird es meine Aufgabe sein, Schülerinnen und Schüler unter anderem in die Grundlagen der Informatik einzuführen. Dazu zählt nicht zuletzt die Boolesche Logik, welche den theoretischen Unterbau der Informatik bildet. Für den normalen Benutzer ist im Computeralltag von Booleschen Funktionen oder Logikgattern nichts mehr zu sehen. Umso wichtiger finde ich es daher, deren fundamentale Bedeutung für die Informatik im Schulunterricht hervor zu streichen und sichtbar zu machen. Wenngleich die Realisierung eines vollständigen Computersystems mittels Gatterschaltungen viel zu komplex für den Unterricht ist, können durchaus einfache Prinzipien veranschaulicht werden. So soll ein grundlegendes Verständnis für die Arbeitsweise von Computersystemen oder zumindest eines Taschenrechners vermittelt werden. Gleich am Anfang dieses Themas kann die Boolesche Logik stehen. Nach diesem Konzept können die technische Umsetzung (Logikgatter) sowie das binäre Zahlensystem besprochen werden. Auf diese Weise soll SchülerInnen zum Beispiel klar werden, warum Information binär codiert gespeichert wird.

Das Ziel meiner Diplomarbeit war es, ein Computerprogramm zu entwickeln, das den Lehrer im Schulunterricht bei der Vermittlung eines Teils der oben angeführten Konzepte unterstützt. Dabei ist eine Anwendung entstanden, die es ermöglicht, Gatterschaltungen zu konstruieren beziehungsweise zwischen den beiden Darstellungsformen Boolesche Funktion und Logikgatter zu wechseln. Zusätzlich werden einige logische Funktionalitäten zur Verfügung gestellt, sowie die Möglichkeit geboten, Aufgaben für SchülerInnen zu erzeugen, durchzuführen und zu bewerten. Das Programm soll dabei helfen, das Thema zu erschließen, ist jedoch explizit nicht als Lernpfad konzipiert. Die vorliegende Arbeit dokumentiert das entstandene Programm sowie dessen Entwicklung. Abschließend werden noch mögliche Anwendungen des Programms im Unterricht besprochen.

2 Pflichtenheft

Im folgenden Pflichtenheft habe ich zu Beginn meiner Arbeit die zu erreichenden Ziele festgeschrieben. Vor dem Start der Entwicklung wurde es von meinem Betreuer abgenommen. Das Pflichtenheft wurde nach der Gliederung von Balzert[1] verfasst.

2.1 Zielbestimmungen

Es soll eine Software erstellt werden, die es Schülerinnen und Schülern im Unterricht erlaubt, Gatterschaltungen zu konstruieren und verifizieren zu lassen.

2.1.1 Musskriterien

- Der Benutzer kann
    - eine Gatterschaltung bauen.
        - eine vom Computer vorgegebene Boolesche Funktion als Gatterschaltung nachbauen.
    - die Boolesche Funktion zur selbstgebauten Gatterschaltung errechnen lassen.
    - eine Boolesche Funktion für eine bestimmte Belegung verifizieren lassen.
    - eine Boolesche Funktion eingeben und als Gatterschaltung vom Computer nachbauen lassen.
- Das Programm
    - stellt Gatterbausteine mit nationalen Symbolen zur Verfügung.
    - enthält die Booleschen Funktionen „und“, „oder“ und „nicht“.
    - kann zwei verschiedene Eingangswerte für die Gatter verarbeiten.
    - kann eine Boolesche Funktion beziehungsweise eine Gatterschaltung für eine Belegung der Eingangswerte auswerten.
    - kann eine Gatterschaltung und eine Boolesche Funktion auf Äquivalenz prüfen.
    - speichert bei Beendigung den aktuellen Stand des Programms, also Gatterschaltung und Boolesche Funktion.

[1] Balzert, Helmut: Lehrbuch der Software-Technik. Band 1: Spektrum-Akademischer Verlag. Heidelberg 2000, S. 115-118.

- Weiteres
    - Gatterbausteine können zwei Eingangswerte haben.
    - Der Ausgang eines Gatters kann mit einem Eingang eines anderen Gatters verbunden sein.
    - Boolesche Funktionen und Gatterschaltungen beinhalten maximal zwei Argumente.

2.1.2 Wunschkriterien

- Das Programm
    - enthält weitere Boolesche Funktionen: NAND, NOR, XOR, Implikation, Äquivalenz.
    - kann Boolesche Funktionen und Gatterschaltungen mit beliebig vielen Argumenten enthalten.
    - evaluiert die Kenntnisse des Benutzers in Form von Punkten.
    - speichert die Ergebnisse des Benutzers als Datei.
    - lädt Aufgaben (Boolesche Funktionen) als Datei.
    - speichert Gatterschaltungen als Bilddatei, wahlweise mit und ohne Boolescher Funktion.
    - stellt Gatter wahlweise mit nationalen oder internationalen Symbolen dar.

2.1.3 Abgrenzungskriterien

- Das Programm soll Schülerinnen und Schüler nicht die Theorie Boolescher Algebra und Gatterschaltungen lehren, ist also nicht als Lernpfad konzipiert.

2.2 Produkteinsatz

2.2.1 Anwendungsbereiche

Die Software soll im schulischen Bereich, speziell im Informatikunterricht angewendet werden. Das Programm soll Schülerinnen und Schülern die Möglichkeit geben, das Bauen von Gatterschaltungen zu erlernen und zu üben.

2.2.2 Zielgruppen

Das Programm soll von Schülerinnen und Schülern des Informatikunterrichts verwendet werden. Da Informatik in der fünften Klasse AHS beziehungsweise in unterschiedlichen Klassen in den BHS als Pflichtfach unterrichtet wird, sind

Schülerinnen und Schüler der Oberstufe, vor allem im Alter von etwa 15 Jahren, Zielgruppe.
Vorausgesetzt werden Basiskenntnisse im Umgang mit dem Computer, sowie die theoretischen Grundlagen Boolescher Algebra.

2.2.3 Betriebsbedingungen

Das Programm wird voraussichtlich nur während des Informatikunterrichts in der Schule, eventuell auch zu Hause verwendet werden. Daher läuft das Programm vermutlich jeweils nur kurz, also weniger als 2 Stunden. Daher sollte der Betrieb des Programms keine Betreuung durch einen Administrator erfordern.

2.3 Produktumgebung

Das Programm wird mit Java geschrieben und läuft daher auf unterschiedlichen Plattformen. Auf jeden Fall wird es unter folgenden Voraussetzungen laufen:

2.3.1 Software

- Betriebssystem Windows XP oder höher.
- Java Runtime Environment 1.6 oder höher.

2.3.2 Hardware

- Ein Rechner, auf dem die oben genannte Software läuft.

2.4 Produktfunktionen

Funktionen, die das Programm für den Benutzer leistet:

- *Eine Gatterschaltung zeichnen lassen:* Der Benutzer gibt eine Boolesche Funktion ein und lässt vom Programm die entsprechende Gatterschaltung zeichnen.
- *Eine Boolesche Funktion erstellen lassen:* Der Benutzer zeichnet eine Gatterschaltung und lässt vom Programm die entsprechende Boolesche Funktion anzeigen.
- *Eine Boolesche Funktion eingeben und die äquivalente Gatterschaltung zeichnen:* Der Benutzer gibt eine Boolesche Funktion ein und zeichnet die dazugehörige Gatterschaltung. Das Programm überprüft die Äquivalenz der beiden.

- *Eine Gatterschaltung zu einer Booleschen Funktion zeichnen:* Das Programm erzeugt eine Boolesche Funktion und der Benutzer zeichnet eine äquivalente Gatterschaltung.
- *Eine Boolesche Funktion zu einer Gatterschaltung eingeben:* Das Programm erzeugt eine Gatterschaltung und der Benutzer gibt die dazu äquivalente Boolesche Funktion ein.
- *Eine Boolesche Funktion evaluieren:* Das Programm evaluiert eine Boolesche Funktion für eine vom Benutzer vorgegebene Belegung der Eingangswerte.

Alle Eingaben des Benutzers können vom Programm ausgewertet und Boolesche Funktion und Gatterschaltung auf Äquivalenz getestet werden.

2.5 Produktdaten

Das Produkt speichert vor jeder Beendigung die aktuell angezeigte Boolesche Funktion sowie die aktuelle Gatterschaltung.
In einem späteren Erweiterungsschritt können auch die Punkteergebnisse der Benutzer gespeichert werden.

2.6 Produktleistungen

Laufzeit: Evaluierungen beziehungsweise Prüfungen auf Äquivalenz sollten innerhalb von Sekundenbruchteilen erfolgen.
Akkumulation: Fehlermeldungen, etwa bei falschen Eingaben, werden akkumuliert ausgegeben.
Toleranz: Fehlerhafte Eingaben können von dem Benutzer korrigiert werden.

2.7 Benutzeroberfläche

Die Benutzeroberfläche besteht aus einem einfachen Fenster, welches alle Funktionen beinhaltet, die der Benutzer in Anspruch nehmen kann. Ein größerer Bereich ist für das Zeichnen von Gatterschaltungen vorgesehen. Für die Ein- und Ausgabe Boolescher Funktionen werden zwei getrennte Felder zur Verfügung stehen. Sollten zusätzliche Funktionalitäten es erforderlich machen, wird zum Erhalt der Übersichtlichkeit ein Optionsmenü integriert.
Das Zeichnen von Gattern wird mittels Drag-and-Drop realisiert. Für die Eingabe von Booleschen Operatoren in der Eingabezeile wird eventuell eine eigene

Tastenbelegung notwendig sein. Alternativ können Buttons für die einzelnen Symbole verwendet werden.
Die Oberfläche wird mit Java Swing beziehungsweise Java2D gestaltet.

2.8 Qualitätsbestimmungen

	Sehr wichtig	Wichtig	Normal	Unwichtig
Korrektheit	X			
Robustheit		X		
Zuverlässigkeit		X		
Benutzerfreundlichkeit		X		
Effizienz		X		
Kompatibilität			X	
Portierbarkeit			X	
Erweiterbar			X	

Entsprechend dem Einsatz in der Schule wird auf die Entwicklung einer einfachen, robusten und benutzerfreundlichen Software Wert gelegt. Besonders wichtig ist selbstverständlich die Korrektheit des Programms.

2.9 Entwicklungsumgebung

2.9.1 Software und Tools

- Windows 7
- JDK 6
- Eclipse 3.6

2.9.2 Hardware

- Laptop mit aktueller Ausstattung und 1900x1080 Pixel Auflösung.

3. Anwendungsfälle

3.1 Anwendungsfalldiagramm

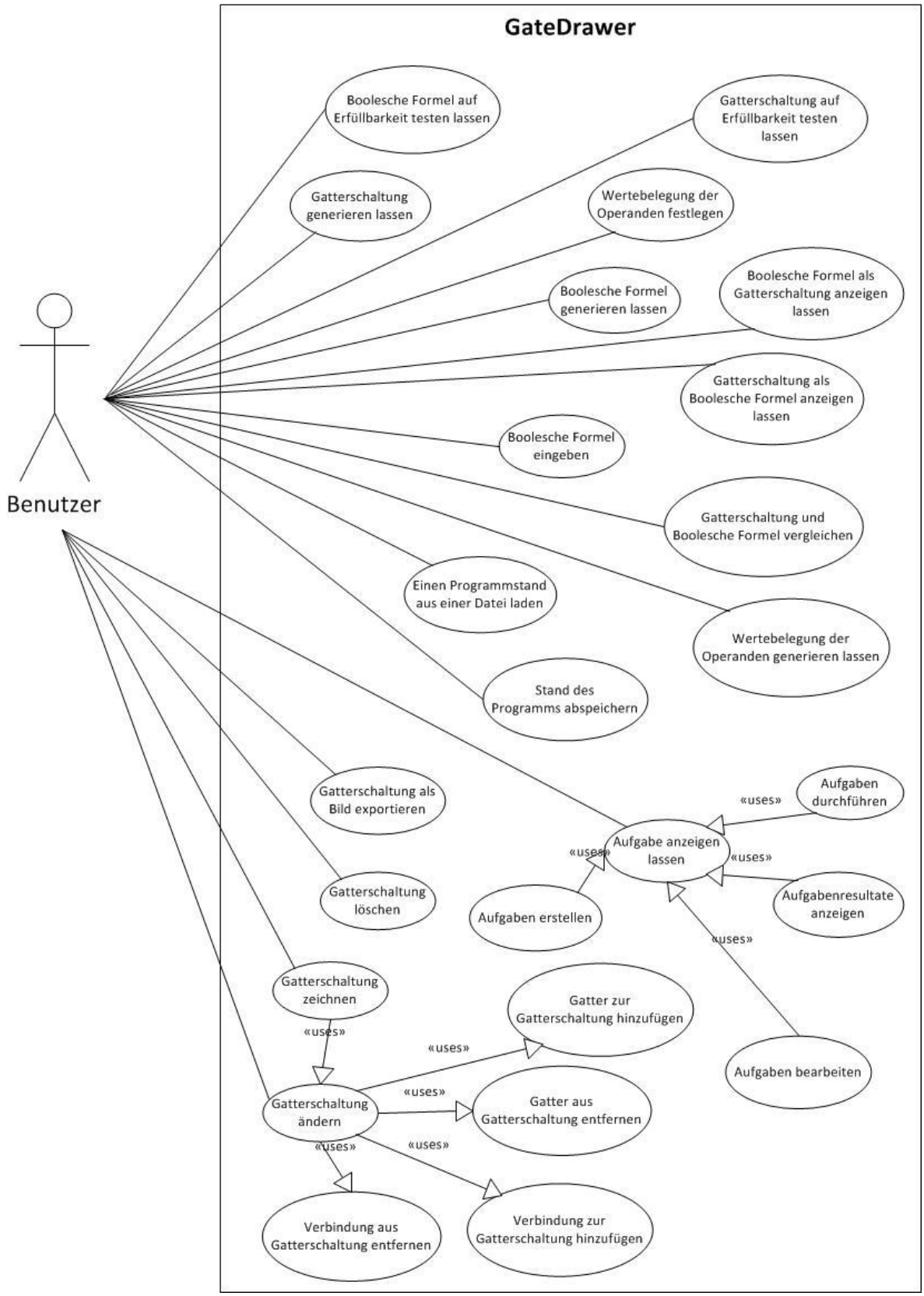


Abb.1: Anwendungsfalldiagramm

3.2 Anwendungsfallbeschreibungen

1. Gatterschaltung zeichnen

- **Kurzbeschreibung**: Der User zeichnet eine neue Gatterschaltung.
- **Scenarios**:

  Hauptszenario: Der User zeichnet eine neue Gatterschaltung. Dazu stehen Symbole für die unterschiedlichen Elemente, wie Gatter, Operanden und Verbindungen, zur Verfügung.

  Mit einem Mausklick positioniert der User ein Element auf der Zeichenebene.

  Vorbedingung: Es wurde keine Aktion ausgeführt, die zu einer Sperre der Zeichenebene führt.

  Nachbedingung: Die gezeichnete Gatterschaltung wird angezeigt. Der Wahrheitswert der Gatterschaltung für die aktuelle Wertebelegung der Operanden wird angezeigt.
- **Includes**: Gatterschaltung ändern.

2. Gatterschaltung ändern

- **Kurzbeschreibung**: Der User ändert eine bereits vorhandene Gatterschaltung.
- **Scenarios**:

  Hauptszenario: Der User ändert eine bestehende Gatterschaltung. Dazu stehen Symbole für die unterschiedlichen Elemente, wie Gatter, Operanden und Verbindungen, zur Verfügung. Mit einem Mausklick positioniert der User ein Element auf der Zeichenebene. Mittels Drag and Drop können Elemente neu positioniert werden.

  Vorbedingung: Es wurde keine Aktion ausgeführt, die zu einer Sperre der Zeichenebene führt. Eine bestehende Schaltung wird angezeigt.

  Nachbedingung: Die geänderte Schaltung wird angezeigt. Der Wahrheitswert der Gatterschaltung für die aktuelle Wertebelegung der Operanden wird angezeigt.
- **Includes:** Gatter zur Gatterschaltung hinzufügen, Gatter aus Gatterschaltung entfernen, Verbindung zur Gatterschaltung hinzufügen, Verbindung aus Gatterschaltung entfernen.

3. **Gatter zur Gatterschaltung hinzufügen**
   - **Kurzbeschreibung**: Der User fügt ein Gatter zu einer bestehenden Gatterschaltung hinzu.
   - **Scenarios**:

     Hauptszenario: Der User wählt ein Symbol für ein Gatter aus und klickt in die Zeichenebene. An dieser Stelle wird ein neues Gatter eingefügt.

     Fehlerszenario: Der User klickt auf ein anderes Gatter oder eine Verbindung. In diesem Fall wird kein neues Gatter eingefügt.

     Vorbedingung: Es wurde keine Aktion ausgeführt, die zu einer Sperre der Zeichenebene führt.

     Nachbedingung: Eine eventuell bereits vorhandene Gatterschaltung, sowie das neue Gatter werden angezeigt.
4. **Gatter aus Gatterschaltung entfernen**
   - **Kurzbeschreibung**: Der User entfernt ein Gatter aus einer bestehenden Gatterschaltung.
   - **Scenarios**:

     Hauptszenario: Der User klickt rechts auf ein Gatter in einer Gatterschaltung oder auf eine bestehende Selektion und wählt „entfernen“ aus. Alle markierten Gatter sowie ihre Verbindungen zu anderen Gattern werden gelöscht.

     Alternativszenario: Alternativ kann der User die „DEL“-Taste drücken, was zum selben Ergebnis führt.

     Vorbedingung: Es wurde keine Aktion ausgeführt, die zu einer Sperre der Zeichenebene führt. Mindestens ein Gatter befindet sich in der Zeichenebene.

     Nachbedingung: Eine eventuell bereits vorhandene Gatterschaltung wird ohne das/die entfernte/n Gatter angezeigt.
5. **Verbindung zur Gatterschaltung hinzufügen**
   - **Kurzbeschreibung**: Der User fügt eine Verbindung zu einer bestehenden Gatterschaltung hinzu.
   - **Scenarios**:

     Hauptszenario: Der User wählt das Symbol für „Verbindung“ aus und klickt ein Start- und ein End-Gatter in der Zeichenebene an. Zwischen den beiden Gattern entsteht eine Verbindung.

     Fehlerszenario: Der User klickt nacheinander auf zwei Gatter-Eingänge oder zwei Gatter-Ausgänge. In diesem Fall wird keine neue Verbindung gezeichnet.

Vorbedingung: Es wurde keine Aktion ausgeführt, die zu einer Sperre der Zeichenebene führt. Mindestens zwei Gatter befinden sich in der Zeichenebene.

Nachbedingung: Eine eventuell bereits vorhandene Gatterschaltung, sowie die neue Verbindung werden angezeigt.

6. **Verbindung aus Gatterschaltung entfernen**
   - **Kurzbeschreibung**: Der User entfernt eine Verbindung aus einer bestehenden Gatterschaltung.
   - **Scenarios**:

     Hauptszenario: Der User klickt rechts auf eine Verbindung in einer Gatterschaltung und wählt „entfernen“ aus. Die Verbindung wird gelöscht, die verbunden Gatter bleiben jedoch bestehen. Alle markierten Gatter sowie ihre Verbindungen zu anderen Gattern werden gelöscht.

     Alternativszenario: Alternativ kann der User das „Verbindung löschen“ Symbol auswählen und dann auf eine Verbindung klicken, was zum selben Ergebnis führt.

     Vorbedingung: Es wurde keine Aktion ausgeführt, die zu einer Sperre der Zeichenebene führt. Die angezeigte Gatterschaltung enthält mindestens eine Verbindung.

     Nachbedingung: Eine eventuell bereits vorhandene Gatterschaltung wird ohne die entfernte Verbindung angezeigt.
7. **Gatterschaltung löschen**
   - **Kurzbeschreibung**: Der User löscht eine ganze bestehende Gatterschaltung.
   - **Scenarios**:

     Hauptszenario: Der User klickt auf das „Löschen“ Symbol oder markiert den gesamten Bereich und löscht den markierten Bereich. Alle Gatter und Verbindungen werden gelöscht.

     Vorbedingung: Es wurde keine Aktion ausgeführt, die zu einer Sperre der Zeichenebene führt.

     Nachbedingung: Es wird eine leere Zeichenebene angezeigt.
8. **Boolesche Funktion eingeben**
   - **Kurzbeschreibung**: Der User gibt eine Boolesche Funktion ein.

- **Scenarios**:

  Hauptszenario: Der User gibt in ein dafür vorgesehenes Feld eine Boolesche Funktion ein. Für die verschiedenen logischen Operatoren stehen eigene Tasten(kombinationen) zur Verfügung. Das System zeigt den Wahrheitswert.

  Vorbedingung: Es wurde keine Aktion ausgeführt, die zu einer Sperre der Zeichenebene führt.

  Nachbedingung: Es wird eine leere Zeichenebene angezeigt.

9. Gatterschaltung generieren lassen

- **Kurzbeschreibung**: Der User lässt vom System eine Gatterschaltung generieren.
- **Scenarios**:

  Hauptszenario: Per Klick lässt der User das System eine neue, zufällige Gatterschaltung generieren.

  Vorbedingung: Es wurde keine Aktion ausgeführt, die zu einer Sperre der Zeichenebene führt.

  Nachbedingung: Die generierte Schaltung wird angezeigt. Der Wahrheitswert der Gatterschaltung für die aktuelle Wertebelegung der Operanden wird angezeigt.
- **Nonfunctional Constraints**: Schaltungen müssen zufällig aus den verfügbaren Operatoren und Operanden generiert werden. Eine bereits generierte Schaltung darf während der Laufzeit nicht noch einmal vorkommen. Die generierten Schaltungen sollen in ihrem Umfang, das heißt der Anzahl der Operatoren und Operanden, variieren.

10. Boolesche Funktion generieren lassen

- **Kurzbeschreibung**: Der User lässt vom System eine Boolesche Funktion generieren.
- **Scenarios**:

  Hauptszenario: Per Klick lässt der User das System eine neue, zufällige Boolesche Funktion generieren.

  Vorbedingung: Es wurde keine Aktion ausgeführt, die zu einer Sperre der Funktionseingabe führt.

  Nachbedingung: Die generierte Boolesche Funktion wird angezeigt. Der Wahrheitswert der Booleschen Funktion für die aktuelle Wertebelegung der Operanden wird angezeigt.

- **Nonfunctional Constraints**: Boolesche Funktionen müssen zufällig aus den verfügbaren Operatoren und Operanden generiert werden. Eine bereits generierte Funktion darf während der Laufzeit nicht noch einmal vorkommen. Die generierten Funktionen sollen in ihrem Umfang, das heißt der Anzahl der Operatoren und Operanden, variieren.

11. Gatterschaltung und Boolesche Funktion vergleichen lassen

- **Kurzbeschreibung**: Der User lässt das System eine Gatterschaltung mit einer Booleschen Funktion vergleichen.
- **Scenarios**:

  Hauptszenario: Per Klick lässt der User das System eine vorhandene Gatterschaltung mit einer Booleschen Funktion vergleichen. Das System zeigt an, ob die beiden äquivalent sind. Es ist dabei egal, ob Funktion und Gatterschaltung per Hand eingegeben, vom System generiert oder aus einer Datei geladen worden sind.

  Fehlerszenario: Falls keine Funktion und/oder keine Gatterschaltung vorhanden sind, wird „falsch“ ausgegeben.

  Nachbedingung: Das System zeigt an, ob Funktion und Gatterschaltung äquivalent sind.

12. Wertebelegung der Operanden festlegen

- **Kurzbeschreibung**: Der User legt die Wahrheitswerte für die verwendeten Operanden fest.
- **Scenarios**:

  Hauptszenario: Per Klick legt der User für jeden einzelnen Operanden einen Wahrheitswert fest. Dieser Wahrheitswert gilt global für jeden Operanden dieses Namens. Gatterschaltung und Boolesche Funktion werden nach jeder Änderung automatisch neu evaluiert.

  Nachbedingung: Das System zeigt alle Wahrheitswerte der Operanden sowie für die Gatterschaltung und die Funktion an.

13. Wertebelegung der Operanden generieren lassen

- **Kurzbeschreibung**: Der User lässt das System eine zufällige Wertebelegung für die verwendeten Operanden erzeugen.

- **Scenarios**:

  Hauptszenario: Per Klick erzeugt das System zufällige Wahrheitswerte für alle Operanden und zeigt diese an. Boolesche Funktion und Gatterschaltungen werden anschließend neu evaluiert.

  Nachbedingung: Das System zeigt alle Wahrheitswerte der Operanden sowie für die Gatterschaltung und die Funktion an.

14. Boolesche Funktion als Gatterschaltung anzeigen lassen

- **Kurzbeschreibung**: Der User lässt vom System die zu einer eingegebenen Booleschen Funktion äquivalente Gatterschaltung generieren und anzeigen.
- **Scenarios**:

  Hauptszenario: Per Klick erzeugt das System eine äquivalente Gatterschaltung und zeigt diese an.

  Nachbedingung: Das System zeigt die generierte Gatterschaltung sowie ihren Wahrheitswert an.

15. Gatterschaltung als Boolesche Funktion anzeigen lassen

- **Kurzbeschreibung**: Der User lässt vom System die zu einer Gatterschaltung äquivalente Boolesche Funktion generieren und anzeigen.
- **Scenarios**:

  Hauptszenario: Per Klick erzeugt das System eine äquivalente Boolesche Funktion und zeigt diese an.

  Fehlerszenario: Entspricht der Gatterschaltung keine gültige Boolesche Funktion, wird die Funktion so weit wie möglich konstruiert und angezeigt.

  Nachbedingung: Das System zeigt die generierte Funktion sowie ihren Wahrheitswert an.

16. Boolesche Funktion auf Erfüllbarkeit testen lassen

- **Kurzbeschreibung**: Der User lässt das System die Erfüllbarkeit einer Booleschen Funktion überprüfen. Ist die Funktion leer, wird falsch zurückgegeben.
- **Scenarios**:

  Hauptszenario: Per Klick überprüft das System, ob eine Boolesche Funktion erfüllbar ist. Das Ergebnis wird dem User angezeigt.

  Nachbedingung: Das System zeigt an, ob die Funktion erfüllbar ist.

17. **Gatterschaltung auf Erfüllbarkeit testen lassen**
    - **Kurzbeschreibung**: Der User lässt das System die Erfüllbarkeit einer Gatterschaltung überprüfen. Ist die Gatterschaltung leer, wird falsch zurückgegeben.
    - **Scenarios**:

      Hauptszenario: Per Klick überprüft das System, ob eine Gatterschaltung erfüllbar ist. Das Ergebnis wird dem User angezeigt.

      Nachbedingung: Das System zeigt an, ob die Gatterschaltung erfüllbar ist.
18. **Stand des Programms abspeichern**
    - **Kurzbeschreibung**: Der User lässt den aktuellen Stand des Systems abspeichern.
    - **Scenarios**:

      Hauptszenario: In einem eigenen Fenster wählt der User eine Datei aus, in welche der aktuelle Programmstand gespeichert werden soll. Ist die Datei noch nicht vorhanden, wird sie vom Programm neu angelegt. Gespeichert werden die aktuell angezeigte Gatterschaltung sowie die eingegebene Funktion.

      Alternativszenario: Der User hat eine bereits existierende Datei ausgewählt. In diesem Fall wird nachgefragt, ob die Datei überschrieben werden soll. Stimmt der User zu, wird fortgefahren. Ansonsten wird das Speichern abgebrochen.

      Fehlerszenario: Tritt ein unvorhergesehener Fehler beim Schreiben in die Datei auf, wird der User davon benachrichtigt.

      Nachbedingung: Der aktuelle Programmstand wurde in die vom User spezifizierte Datei gespeichert.
19. **Einen Programmstand aus einer Datei laden**
    - **Kurzbeschreibung**: Der User lässt einen Programmstand aus einer Datei laden.
    - **Scenarios**:

      Hauptszenario: In einem eigenen Fenster wählt der User eine Datei aus, aus welcher der neue Programmstand geladen werden soll. Geladen werden eine Gatterschaltung sowie eine Boolesche Funktion. Der alte Programmstand wird gelöscht.

Alternativszenario: Der User wählt keine Datei aus oder bricht die Auswahl ab. In diesem Fall wird kein Programmstand geladen. Der alte Programmstand bleibt erhalten.

Fehlerszenario: Tritt ein unvorhergesehener Fehler beim Lesen der Datei auf, wird der User davon benachrichtigt.

Nachbedingung: Der neue Programmstand wird angezeigt, der alte ist gelöscht.

20. Gatterschaltung als Bild exportieren

- **Kurzbeschreibung**: Der User lässt die aktuelle Gatterschaltung als Bild exportieren.
- **Scenarios**:

  Hauptszenario: In einem eigenen Fenster wählt der User eine Datei aus, in welche das Bild gespeichert werden soll. Ist die Datei noch nicht vorhanden, wird sie vom Programm neu angelegt. Das System generiert ein Bild aus der Gatterschaltung und speichert dieses in der Datei ab.

  Alternativszenario: Der User wählt keine Datei aus oder bricht die Auswahl ab. In diesem Fall wird kein Bild erzeugt und exportiert.

  Fehlerszenario: Tritt ein unvorhergesehener Fehler beim Schreiben der Datei auf, wird der User davon benachrichtigt.

  Fehlerszenario: Es existiert keine Gatterschaltung, die exportiert werden könnte. Dem User wird eine entsprechende Nachricht angezeigt und der Vorgang wird beendet.

  Nachbedingung: Die aktuelle Gatterschaltung wurde als Bild exportiert.

21. Aufgaben erstellen

- **Kurzbeschreibung**: Der User erstellt ein Set von Aufgaben.
- **Scenarios**:

  Hauptszenario: Der User wählt „Aufgaben erstellen“ aus. Eine Toolbar zum Erstellen von Aufgaben wird dem User angezeigt. Er hat die Möglichkeit, mehrere Aufgaben unterschiedlicher Art zu erstellen. Jeder Aufgabe können Punkte und eine Anzahl an Versuchen zugewiesen werden. Am Schluss werden die Aufgaben in einer Datei gespeichert, die der User auswählt.

  Alternativszenario: Der User verwirft die Aufgaben und speichert sie nicht ab. Die Toolbar wird entfernt und die kreierten Aufgaben werden verworfen.

  Fehlerszenario: Tritt ein unvorhergesehener Fehler beim Schreiben der Datei auf, wird der User davon benachrichtigt.

Nachbedingung: Die kreierten Aufgaben wurden in eine Datei gespeichert.

- **Includes**: Aufgabe anzeigen lassen.

22. Aufgaben bearbeiten

- **Kurzbeschreibung**: Der User bearbeitet ein Set von Aufgaben.
- **Scenarios**:

  Hauptszenario: Der User wählt „Aufgaben bearbeiten“ aus. Aus einem eigenen Fenster wählt er eine Datei aus, in der Aufgaben gespeichert sind, die er bearbeiten möchte. Eine Toolbar zum Bearbeiten von Aufgaben wird angezeigt. Es besteht die Möglichkeit, Aufgaben neu hinzuzufügen, bestehende Aufgaben zu löschen oder zu ändern. Am Schluss werden die Aufgaben in einer Datei gespeichert, die der User auswählt.

  Alternativszenario: Der User verwirft die Aufgaben und speichert sie nicht ab. Die Toolbar wird entfernt und die kreierten Aufgaben werden verworfen.

  Fehlerszenario: Tritt ein unvorhergesehener Fehler beim Schreiben der Datei auf, wird der User davon benachrichtigt.

  Nachbedingung: Die kreierten Aufgaben wurden in eine Datei gespeichert.
- **Includes**: Aufgabe anzeigen lassen.

23. Aufgaben durchführen

- **Kurzbeschreibung**: Der User führt ein Set von Aufgaben aus.
- **Scenarios**:

  Hauptszenario: Der User wählt „Aufgaben ausführen“ aus. Aus einem eigenen Fenster wählt er eine Datei aus, in der Aufgaben gespeichert sind, die er ausführen möchte. Eine Toolbar zum Durchführen von Aufgaben wird angezeigt. Jede Aufgabe wird einzeln angezeigt und durchgeführt. Der User kann seine Lösung überprüfen und bewerten lassen. Jede Aufgabe kann nur eine begrenzte Anzahl oft durchgeführt werden. Der User kann eine Aufgabe auch aufgeben und sich die korrekte Lösung anzeigen lassen. Am Schluss werden die Aufgaben in derselben Datei gespeichert, aus der sie geladen wurden.

  Fehlerszenario: Tritt ein unvorhergesehener Fehler beim Schreiben der Datei auf, wird der User davon benachrichtigt.

  Fehlerszenario: Die Datei, aus welcher die Aufgaben geladen worden sind, ist beim Speichern nicht mehr vorhanden. Der User kann eine neue Datei auswählen, in welche gespeichert wird.

Nachbedingung: Die Aufgaben wurden mitsamt den Resultaten in eine Datei gespeichert.

- **Includes**: Aufgabe anzeigen lassen.

24. Aufgabenresultate anzeigen

- **Kurzbeschreibung**: Der User lässt sich Aufgabenresultate anzeigen.
- **Scenarios**:

  Hauptszenario: Der User wählt „Resultate anzeigen“ aus. Aus einem eigenen Fenster wählt er eine Datei aus, in der Aufgaben und Resultate gespeichert sind, die er anzeigen möchte. Eine Toolbar zum Anzeigen der Resultate wird angezeigt. Jede Aufgabe wird mit ihren Resultaten einzeln angezeigt. Außerdem wird eine Gesamtstatistik angezeigt, wie viele Aufgaben versucht und gelöst und wie viele Punkte erreicht worden sind.

  Fehlerszenario: Tritt ein unvorhergesehener Fehler beim Lesen der Datei auf, wird der User davon benachrichtigt.
- **Includes**: Aufgabe anzeigen lassen.

25. Aufgabe anzeigen lassen

- **Kurzbeschreibung**: Eine Aufgabe aus einem Set von Aufgaben wird angezeigt.
- **Scenarios**:

  Hauptszenario: Eine Aufgabe wird angezeigt. Dazu gehört eine Gatterschaltung und/oder eine Boolesche Funktion. Außerdem wird ein Aufgabentext angezeigt, der die jeweilige Aufgabe beschreibt. Ist eine Aufgabe von einem User gelöst worden, so wird auch die dazu gespeicherte Lösung oder der Lösungsversuch angezeigt.

  Trigger: Aufgaben werden erstellt oder durchgeführt oder Resultate angezeigt.

  Vorbedingung: Ein Set aus zumindest einer Aufgabe ist im Programm vorhanden.

  Nachbedingung: Die gewählte Aufgabe wird angezeigt.

4. Programmstruktur

Für die grundlegende Struktur des Programms wurden drei gängige Softwareentwurfsmuster verwendet, das „Data Access Object“ Muster („DAO Pattern“), das „Model View Controller“ Muster („MVC Pattern“) sowie das „Transfer Object“ Muster. DAOs wurden als Schnittstelle zum Dateisystem eingesetzt, um Programmstände und Aufgaben aus Dateien einzulesen oder in Dateien zu speichern. Außerdem wurde ein DAO verwendet, um Gatterschaltungen als Bilder zu exportieren. Gatterschaltungen setzen sich im Programm zum einen aus Gattern und zum anderen aus Verbindungslinien zusammen. Diese werden getrennt gespeichert und bearbeitet. Um sie in einem einzelnen Schritt übertragen zu können, werden sie gemeinsam in ein Datentransferobjekt gespeichert. Zur Kommunikation mit einem DAO wird ebenfalls ein Transferobjekt verwendet, in dem zusätzlich zur Schaltung auch zu speichernde oder zu ladende Boolesche Funktionen enthalten sind. Das MVC Pattern prägt die Grobstruktur des Programmes. Es ist unterteilt in GUI Klassen, die der Interaktion mit dem Benutzer dienen, Serviceklassen, welche die Programmlogik enthalten, sowie Domainobjekte, welche die Daten über Schaltungen und Aufgaben enthalten. Die folgenden Klassendiagramme veranschaulichen die Verwendung von Entwurfsmustern im Programm. Im Sinne einer besseren Lesbarkeit wurde auf jegliche Beschreibung der Klassen verzichtet.

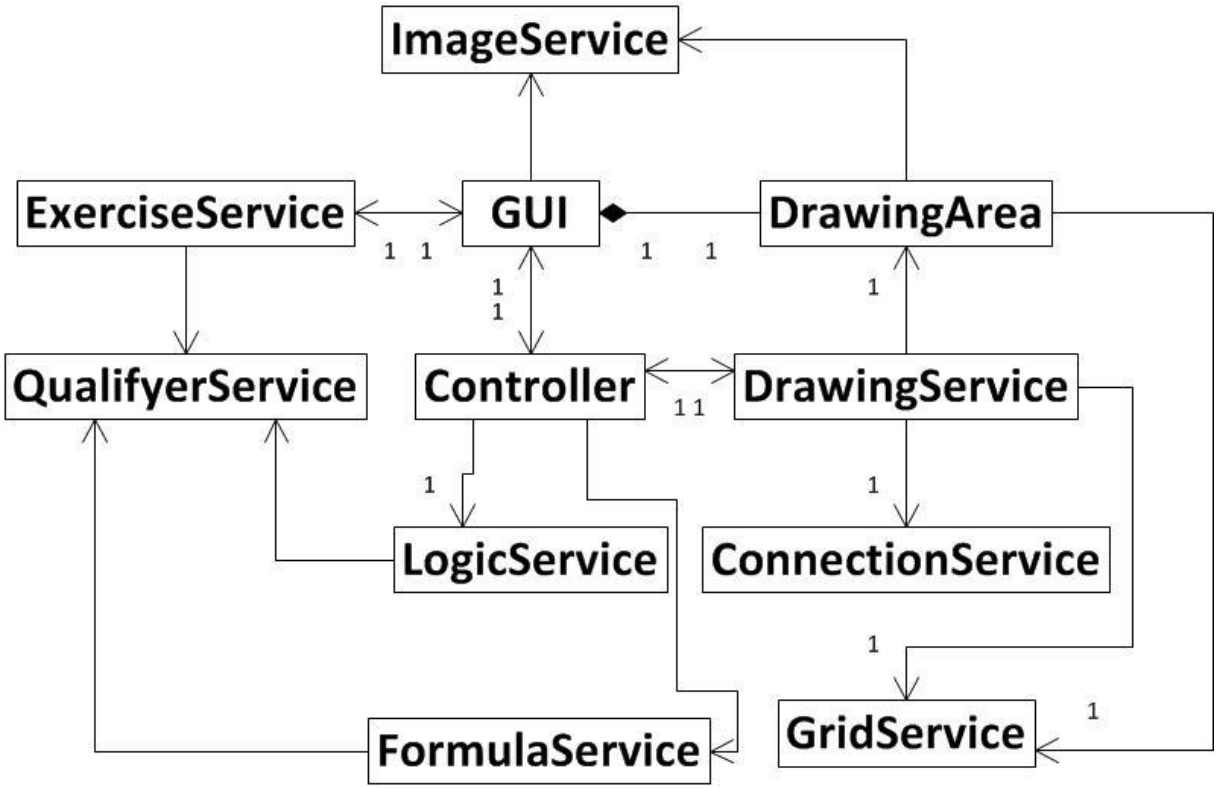


Abb.2: Umsetzung des MVC Musters im Programm.

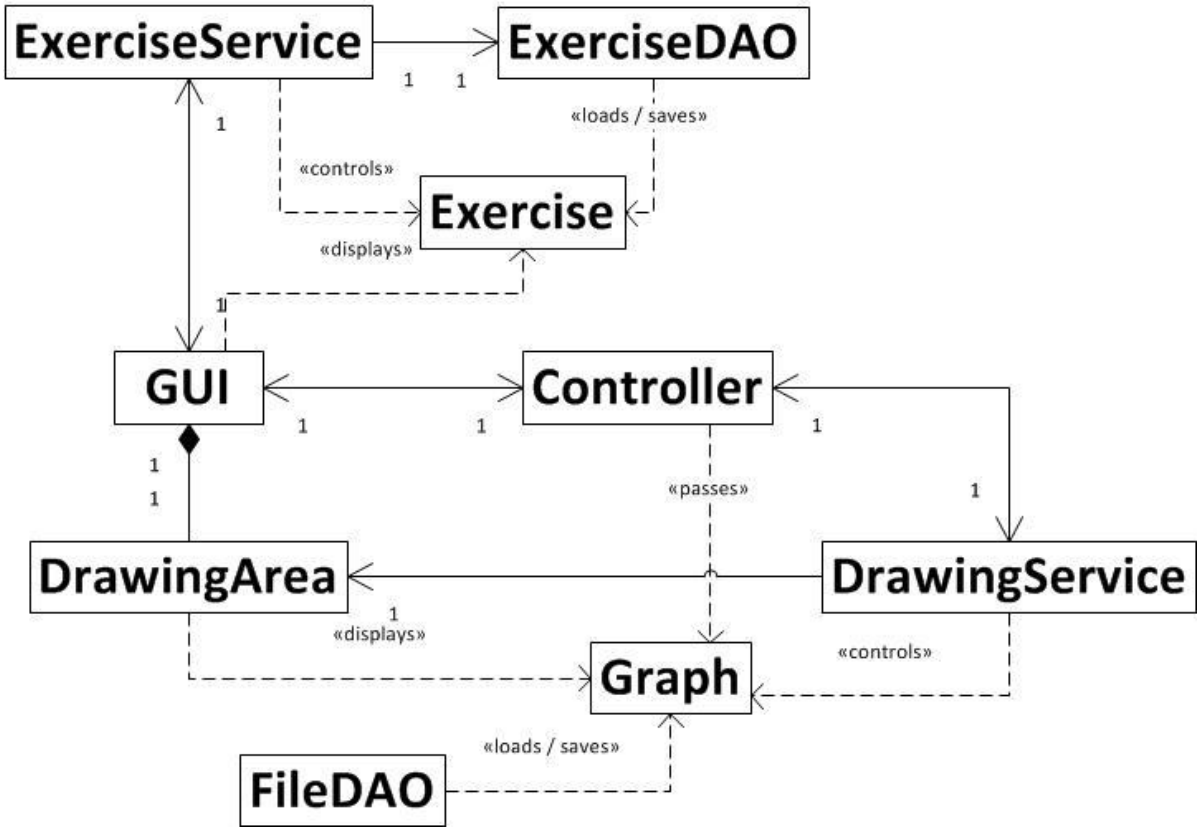


Abb.3: Umsetzung des DAO Musters im Programm.

Das folgende Klassendiagramm zeigt die Datentransferobjekte, die im Paket domain.gui zusammengefasst sind. Aus diesen Elementen setzt sich eine Gatterschaltung zusammen.

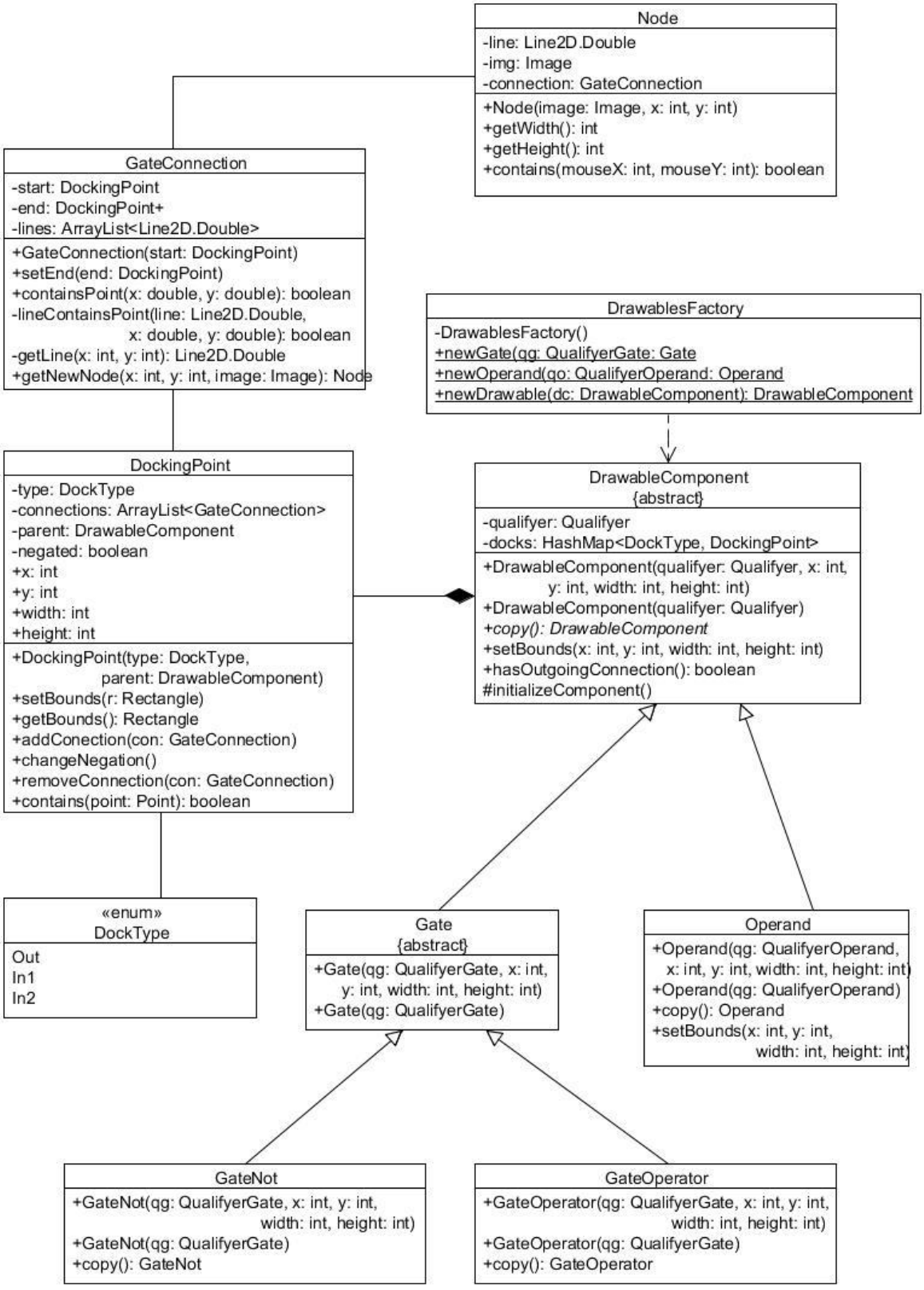


Abb.4: Klassendiagramm des Pakets domain.gui.

5. Systemvoraussetzungen und Lizenzbedingungen

Das Programm läuft unter allen Systemen, auf denen Java Runtime Environment ab Version 6 installiert ist. Wie Java selbst ist das Programm auf jedem Betriebssystem lauffähig. An Hardware wird ein Bildschirm mit einer Auflösung von mindestens 1024x768 Pixeln benötigt.

Das Programm wird unter der GNU General Public License veröffentlicht[2]. Die Veröffentlichung des Programmes erfolgt ohne irgendeine Garantie, sogar ohne die implizite Garantie der Marktreife oder der Verwendbarkeit für einen bestimmten Zweck[3].

[2] Free Software Foundation: GNU General Public Licence. Version 3. 29.6.2007. http://www.gnu.org/licenses/gpl-3.0.html abgerufen am 1.3.2012.

[3] Nach Gerwinski, Peter: GNU General Public Licence. Deutsche Übersetzung der Version 3. 29.6.2007. http://www.gnu.de/documents/gpl.de.html abgerufen am 1.3.2012.

6. Benutzerhandbuch

6.1 Programmstart

Das Programm wird mit der mitgelieferten .jar-Datei gestartet. Für die Eingabeaufforderung lautet der Befehl:

javaw –jar GateDrawer.jar

Nach dem Start wird der vor der letzten Beendigung automatisch gespeicherte Programmstand angezeigt. Wird kein Speicherstand gefunden, wird eine leere Zeichenebene angezeigt.

6.2 Übersicht

Das Programm besteht aus den folgenden Bereichen (vgl. auch Abbildung):

(A) Zeichenbereich

Der Zeichenbereich dient der graphischen Darstellung von Gatterschaltungen. Auf ihm werden vom User Gatterschaltungen gezeichnet oder vom Programm generiert.

(B) Toolbox

In diesem Bereich können Elemente für den Bau einer Gatterschaltung ausgewählt werden. Außerdem werden hier die Wertebelegungen für Operanden festgelegt und Aktionen gestartet.

(C) Textausgabe

In diesem Feld gibt das Programm Resultate logischer Operationen sowie Anweisungen für Aufgaben aus.

(D) Bereich für Benutzereingaben

In diesem Bereich kann der Benutzer Boolesche Funktionen eingeben, generieren und als Gatterschaltung darstellen lassen.

(E) Bereich für Gatterlogik

Dieser Bereich ist mit der dargestellten Gatterschaltung verknüpft. Hier werden Aktionen gestartet, die mit der Gatterschaltung zu tun haben.

(F) Menüleiste

Die Menüleiste stellt grundlegende Programmfunktionen wie etwa Speichern oder Beenden zur Verfügung.

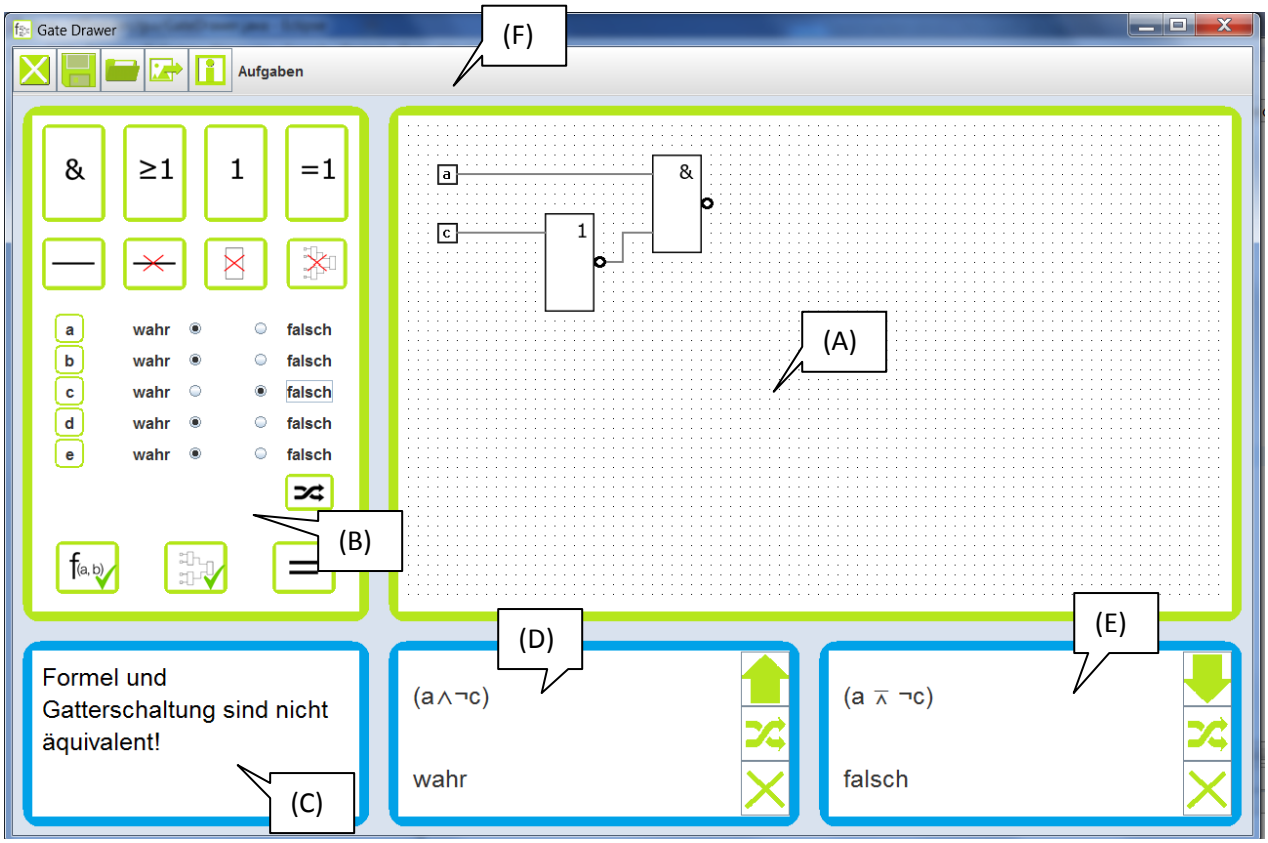


Abb.5: Screenshot des Programms

6.3 Grundfunktionen

6.3.1 Gatterschaltung zeichnen

Für das Zeichnen von Gatterschaltungen steht ein großer Bereich im Programm zur Verfügung (A). Für die Auswahl der Gatterbausteine steht die Toolbox (B) am linken Rand des Programms zur Verfügung.

- Bedienelemente in der Toolbox

  In der Toolbox finden sich die vier grundlegenden Gatterarten, die gezeichnet werden können: AND, OR, NOT und XOR. NAND-, NOR-und XNOR-Gatter können durch Negation von AND, OR und XOR-Gattern erzeugt werden. Darunter befinden sich die Symbole zum Zeichnen von Linien, zum Entfernen von Linien oder Gattern sowie zum Löschen der Schaltung. Als nächstes sind Symbole für die Operanden der Schaltung zu finden. Sie können auf dieselbe Weise verwendet werden, wie die anderen Elemente. Ein gewähltes Element bleibt solange ausgewählt, bis ein anderes Element ausgewählt wird.

- Platzieren von Gatterbausteinen

  Ist ein Baustein in der Toolbox ausgewählt, so kann er mit einem Klick auf eine freie Stelle in der Zeichenebene zur Schaltung hinzugefügt werden. Der Baustein bleibt danach weiterhin selektiert, sodass zusätzliche Elemente desselben Typs durch weitere Klicks hinzugefügt werden können. Die Bausteine richten sich auf der Zeichenebene am sichtbaren Gitter automatisch so aus, dass die linke obere Ecke jeweils genau über einem Gitterknoten liegt.
- Entfernen von Gatterbausteinen

  Gatterbausteine können durch einen Rechtsklick auf das Gatter und anschließende Auswahl der Option „*Gatter entfernen*“ gelöscht werden. Alternativ kann aus der Toolbox das Symbol zum Entfernen von Bausteinen ausgewählt und auf das zu entfernende Gatter geklickt werden. Beim Löschen von Bausteinen werden alle Verbindungen zum Baustein entfernt.
- Erzeugen von Verbindungen

  Verbindungslinien zwischen Gatterbausteinen können mit dem Linien-Symbol aus der Toolbox erzeugt werden. Ist das Element ausgewählt, kann eine Verbindung durch Klicken auf die gewünschten Gatter erzeugt werden. Um einen Ausgang zu selektieren, muss in den rechten Bereich eines Gatters geklickt werden, für einen Eingang auf den linken Bereich. Es ist nicht möglich, zwei Ein- oder Ausgänge miteinander zu verbinden. Um das Zeichnen der Verbindung abzubrechen, kann die Taste *ESC* gedrückt werden.
- Entfernen von Verbindungen

  Gatterverbindungen können durch einen Rechtsklick auf die Verbindung und anschließende Auswahl der Option „*Verbindung entfernen*“ gelöscht werden. Alternativ kann aus der Toolbox das Symbol zum Entfernen von Verbindungen ausgewählt und auf die zu löschende Verbindung geklickt werden.
- Negationen

  Ein- oder Ausgänge von Gatterbausteinen können negiert werden. Ein negierter Ein- oder Aufgang ist an einem kleinen schwarzen Kreis neben dem Gatter zu erkennen. Die Negation kann durch einen Rechtsklick auf den gewünschten Ein- oder Ausgang festgelegt werden.
- Gatterschaltungen editieren

  Eine Gatterschaltung, die in der Zeichenebene angezeigt wird, kann vom Benutzer bearbeitet werden. Wie in (a) beschrieben, können Gatterbausteine

und Linien hinzugefügt oder entfernt werden. Zur Bearbeitung der Gatterschaltung stehen auch die folgenden Funktionalitäten zur Verfügung.

- Drag and Drop
  Gatterbausteine können auf der Zeichenebene mittels Drag and Drop verschoben werden.
- Selektion
  Durch Drücken der linken Maustaste und gleichzeitiges Ziehen der Maus kann ein Bereich auf der Zeichenebene markiert werden. Verschiebe- und Löschaktionen können nun gleichzeitig für alle Elemente im selektierten Bereich durchgeführt werden. Selektierte Elemente können mit der *ENTF*-Taste gelöscht werden. Die Selektion kann durch einen Klick auf die Zeichenebene wieder aufgehoben werden.

6.3.2 Gatterschaltung löschen

Zum Löschen einer ganzen Gatterschaltung steht eine entsprechende Schaltfläche in der Toolbox zur Verfügung. Alternativ kann auch die gesamte Zeichenebene selektiert und gelöscht werden.

6.3.3 Boolesche Funktion eingeben

Es besteht die Möglichkeit, eine Boolesche Funktion in das dafür vorgesehene Textfeld in (D) einzugeben. Für die Eingabe der Operatoren sind die folgenden Tasten vorgesehen:

Tastenkombination	Symbol
“+“	∧ (AND)
“-“	∨ (OR)
“/“	¬ (NOT)
“X“	$\underline{\vee}$ (XOR)
“SHIFT +“	$\overline{\wedge}$ (NAND)
“SHIFT -“	$\overline{\vee}$ (NOR)

Während der Eingabe wird die Funktion laufend evaluiert und ihr Wahrheitswert für die aktuelle Wertebelegung (siehe 4.a.) darunter ausgegeben.

6.3.4 Speichern

Es besteht die Möglichkeit, den jeweils aktuellen Stand des Programms, das heißt die angezeigte Gatterschaltung sowie die eingegebene Boolesche Funktion,

abzuspeichern. Dazu wird das Speichern-Symbol aus der Menüleiste (F) gedrückt und eine Datei ausgewählt. Programmstände werden in Dateien mit der Endung *.gser* gespeichert. Vor jedem Beenden wird automatisch der letzte Stand des Programms gesichert.

6.3.5 Laden

Gespeicherte Programmstände können durch Klicken auf das Laden-Symbol in der Menüleiste und anschließendes Auswählen einer *.gser*-Datei geladen werden. Der geladene Programmstand wird sofort angezeigt. Zu jedem Programmstart wird der letzte Programmstand geladen, so ein solcher verfügbar ist.

6.3.6 Bildexport

Eine Gatterschaltung kann in eine Bilddatei im JPEG-Format exportiert werden. Dafür steht ein entsprechendes Symbol in der Menüleiste zur Verfügung. Die Datei, in welche das Bild exportiert wird, kann ausgewählt werden.

6.4 Logische Funktionen

6.4.1 Wertebelegung

Jedem logischen Operanden kann im Programm ein Wahrheitswert zugewiesen werden. Der zugewiesene Wahrheitswert ist global gültig und beeinflusst den Wahrheitswert einer Funktion oder einer Gatterschaltung. Für Funktion und Gatterschaltung können die Werte nicht getrennt festgelegt werden.

- Manuell
  In der Toolbox können die Wahrheitswerte für jeden Operanden festgelegt werden. Dazu ist neben dem jeweiligen Operanden auf den gewünschten Wahrheitswert zu klicken. Jede Änderung wird automatisch und sofort vom Programm nachvollzogen.
- Zufällig
  Es besteht die Möglichkeit, eine zufällige Wertebelegung generieren zu lassen. Dazu steht ein Button unterhalb der Wertetafel in der Toolbox zur Verfügung.

6.4.2 Automatische Evaluation

Zu jeder Zeit werden die Wahrheitswerte der Booleschen Funktion im Benutzerfeld (D) sowie der Gatterschaltung (im Bereich für Gatterlogik (E)) angezeigt. Sie werden

bei jeder Änderung der Funktion, der Schaltung oder der Wertebelegungen neu evaluiert

6.4.3 Vergleichen

Eine Boolesche Funktion im Benutzerfeld und eine Gatterschaltung können auf Äquivalenz überprüft werden. Dafür steht ein entsprechender Button in der Toolbox zur Verfügung. Das Ergebnis wird in der Textausgabe (C) angezeigt.

6.4.4 Gatterschaltung als Funktion darstellen

Eine Gatterschaltung kann als eine Boolesche Funktion dargestellt werden. Dazu genügt ein Klick auf das Pfeilsymbol im Bereich der Gatterlogik.

6.4.5 Funktion als Gatterschaltung darstellen

Das Programm kann aus einer Booleschen Funktion eine Gatterschaltung generieren. Dafür genügt ein Klick auf das Pfeilsymbol im Benutzerfeld.

6.5.6 Zufällige Funktion oder Gatterschaltung generieren.

Das Programm kann zufällige Boolesche Funktionen oder Gatterschaltungen generieren. Dazu dienen die mittleren Buttons im Benutzerbereich oder in der Gatterlogik. Eine einmal generierte Funktion oder Schaltung wird im selben Programmdurchlauf nicht noch einmal erzeugt.

6.5.7 Erfüllbarkeit überprüfen

Boolesche Funktionen oder Gatterschaltungen können auf ihre Erfüllbarkeit getestet werden. Dafür stehen zwei Buttons in der Toolbox zur Verfügung. Das Ergebnis wird in der Textausgabe angezeigt.

6.5 Aufgaben

Das Programm verfügt über die Möglichkeit, ein Set an Aufgaben zu erzeugen, diese durchzuführen und die Resultate anzuzeigen. Aufgaben werden in Dateien mit der Endung *.gex* gespeichert. Alle Aufgabenoperationen finden sich unter „*Aufgaben*“ in der Menüleiste.

6.5.1 Arten von Aufgaben

Folgende Aufgabentypen stehen zur Auswahl:

- Finde die Funktion: Zu einer Gatterschaltung soll die äquivalente Funktion eingegeben werden.

- Korrigiere die Funktion: Eine Funktion soll so abgeändert werden, dass sie äquivalent zu einer gegebenen Gatterschaltung ist.
- Finde die Gatterschaltung: Zu einer Funktion soll die äquivalente Gatterschaltung konstruiert werden.
- Korrigiere die Gatterschaltung: Eine Gatterschaltung soll so abgeändert werden, dass sie äquivalent zu einer gegebenen Funktion ist.

6.5.2 Aufgaben erzeugen

Zur Erzeugung von Aufgaben ist der entsprechende Eintrag „*Aufgaben kreieren*“ auszuwählen. Die neu erschienene Toolbar stellt die notwendigen Funktionen zur Verfügung:

- *Neue Aufgabe*: Eine neue Aufgabe wird am Ende angelegt. Es kann immer nur eine neue Aufgabe erzeugt werden. Eine weitere Aufgabe kann erst angelegt werden, wenn die letzte Aufgabe gespeichert wurde.
- *Aufgabe speichern*: Die aktuell bearbeitete Aufgabe speichern.
- *Aufgabe löschen*: Die aktuell angezeigte Aufgabe löschen.
- *Punkte*: Angabe, wie viele Punkte die Aufgabe wert ist
- *Versuche*: Angabe, wie viele Versuche ein Benutzer für diese Aufgabe hat.
- *Vor/Zurück*: Anzeige der nächsten/ vorherigen Aufgabe.
- *Abschließen*: Speichern des Aufgabensets in eine Datei.
- *Schließen*: Schließen der Toolbar.

Angezeigt werden außerdem die Anzahl der erzeugten Aufgaben sowie die erreichbare Gesamtpunktezahl.

6.5.3 Aufgaben bearbeiten

Ein Set an Aufgaben kann nach dem Speichern erneut bearbeitet werden. Dazu ist der Eintrag „*Aufgaben bearbeiten*“ auszuwählen. Nun kann eine *.gex*-Datei mit Aufgaben geladen werden. Die weitere Funktionalität ist analog zu der in (b).

6.5.4 Aufgaben durchführen

Es besteht die Möglichkeit, ein Aufgabenset zu lösen. Dazu wird „*Aufgaben lösen*“ aus dem Menü ausgewählt. Die Aufgaben werden aus einer *.gex*-Datei geladen und einzeln angezeigt. Es besteht nun die Möglichkeit, eine passende Funktion einzugeben oder Gatterschaltung zu konstruieren.

- *Überprüfen*: Eine eingegebene Lösung überprüfen lassen. Abgegebene Lösungen werden gespeichert und angezeigt. Wird die Aufgabe gelöst, werden Punkte vergeben, abhängig von der maximalen Punktezahl und der benötigten Versuche.
- *Zeige Lösung*: Die korrekte Lösung für eine Aufgabe wird angezeigt. Es sind keine weiteren Lösungsversuche mehr möglich.
- *Vor/Zurück*: Zeigt die nächste/letzte Aufgabe an.
- *Abschließen*: Schließt das Aufgabenset ab. Die Resultate werden automatisch in die Datei gesichert. Ist die Datei nicht mehr verfügbar, besteht die Möglichkeit, eine andere Datei auszuwählen.

6.5.5 Resultate anzeigen

Die Resultate gelöster Aufgaben können mittels des Eintrags „*Ergebnisse laden*“ angezeigt werden. Aus einer *.gex*-Datei werden die Resultate geladen und folgende Informationen angezeigt:

- *Aufgaben insgesamt*: Die Gesamtzahl der Aufgaben in diesem Set.
- *Punkte erreicht*: Die erreichten Punkte sowie die maximal erreichbaren Punkte.
- *Gelöste Aufgaben*: Die Anzahl der gelösten Aufgaben.
- *Versuchte Aufgaben*: Die Anzahl der vom Benutzer versuchten Aufgaben.

Mit „*vor*“/„*zurück*“ kann zwischen den einzelnen Aufgaben des Sets navigiert werden. Im Programm wird die jeweilige Aufgabe sowie die Lösung (oder der Versuch) angezeigt. Außerdem werden angezeigt:

- Die *Nummer der Aufgabe.*
- *Punkte*: Die erreichten Punkte bei dieser Aufgabe.
- *Versuche*: Die Anzahl an benötigten Versuchen für diese Aufgabe.
- *Gelöst*: Ob die Aufgabe korrekt gelöst worden ist.

Mit „*Schließen*“ wird die Anzeige der Resultate geschlossen und die Toolbar entfernt.

7 Einsatzmöglichkeiten im Unterricht

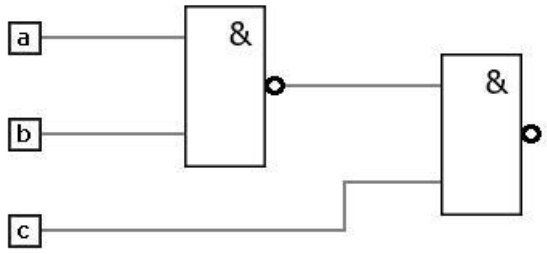
Das Ziel dieser Arbeit war es, ein Computerprogramm für den späteren Einsatz in der Schule zu entwickeln. Entsprechend sollen nun einige Möglichkeiten der Anwendung im Unterricht erläutert werden. Da Informatikunterricht in der AHS nur in der neunten Schulstufe verpflichtend vorgesehen ist, besteht die Zielgruppe aus Oberstufenklasse. Im Lehrplan der 5. Klasse ist vorgesehen, dass Schülerinnen und Schüler

> *„…Einblicke in wesentliche Begriffe und Methoden der Informatik, ihre typischen Denk- und Arbeitsweisen, ihre historische Entwicklung sowie ihre technischen und theoretischen Grundlagen gewinnen und Grundprinzipien von Automaten, Algorithmen und Programmen kennen lernen…"*[4]

sollen. Theoretisch fußt die Informatik wesentlich auf Boolescher Algebra, ohne die Computer nicht denkbar wären. Technisch umgesetzt wird diese binäre Logik durch Gatterschaltungen, deren elementare Funktionalitäten zu den Grundlagen der Informatik gehören. Diese werden im Informatikunterricht häufig vernachlässigt, sollten aber zumindest in der neunten Schulstufe ansatzweise vermittelt werden. Entsprechend zeigt sich das Anwendungsgebiet der entwickelten Software, deren Hauptaugenmerk auf der Darstellung logischer Ausdrücke als Boolesche Funktion oder Gatterschaltung, sowie der Umwandlung zwischen den beiden Formen liegt. Dazu gehört primär das Konstruieren von Logikgattern, aber auch das Herauslesen Boolescher Funktionen aus Schaltungen. Im Unterricht kann das Programm während des gesamten Themenkreises Gatterschaltungen eingesetzt werden. Schülerinnen und Schüler können selbstständig Schaltungen zu Booleschen Funktionen erzeugen. Umgekehrt können auch die Booleschen Funktionen zu gegebenen Gatterschaltungen gefunden werden. Zum einen soll damit die Konstruktion von Gatterschaltungen geübt werden, zum anderen aber auch ein Verständnis dafür vermittelt werden, wie abstrakte, mathematische Logik in einer technischen Konstruktion umgesetzt wird. Dieser Schritt, der im Anschluss an die reine Konstruktionsübung einer bewussten Reflexion bedarf, wird im Unterricht vernachlässigt. Er ist aber essentiell, um den Schülerinnen und Schülern die Motivation für diese ansonsten recht abstrakte Übung bewusst werden zu lassen.

[4] Bundesministerium für Unterricht, Kunst und Kultur (Hg.): Lehrplan Informatik für die AHS Oberstufe. Wien 2004, S. 2.

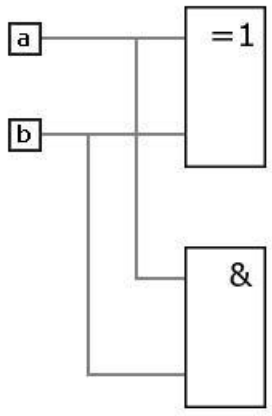
Nachdem die Grundlagen der Konstruktion von Gatterschaltungen geübt worden sind, können schwierigere Aufgaben gestellt werden. Dazu gehört zum Beispiel die Konstruktion einer Schaltung, die nur aus NAND- oder NOR-Gattern besteht und dieselbe Funktion erfüllt wie ein anderer Operator. Als Beispiel wird im Folgenden gezeigt, wie aus NAND-Gattern eine AND-Verknüpfung dargestellt werden kann, wobei der Operand c mit log.1 belegt sein muss.



((a ⊼ b) ⊼ c)

Abb.6: Darstellung eines AND Operators durch NAND-Gatter.

Wird der Stoff vertieft, etwa im Wahlpflichtgegenstand Informatik, können auch komplexere Schaltungen, wie etwa ein Halb- oder Volladdierer dargestellt werden. Allerdings stößt das Programm hier an seine Grenzen, da die Evaluation einer Schaltung mit mehreren Ausgängen nicht möglich ist. Die Konstruktion ist jedoch kein Problem, wie aus Abbildung 7 zu entnehmen ist.



(a ⊻ b)

Abb.7: Ein Halbaddierer, aufgebaut aus XOR- und AND-Gattern.

Möglich sind auch Aufgabenstellungen, welche die Konstruktion einer Schaltung erfordern, die unter einer gegebenen Wertebelegung wahr oder falsch wird. Das

Programm vereinfacht auch die Erstellung einer Wahrheitstabelle durch Schülerinnen und Schüler: Für eine Gatterschaltung oder eine Funktion müssen nur noch alle möglichen Wertebelegungen eingestellt werden, die Wahrheitswerte berechnet das Programm automatisch. Damit erleichtert es auch die Beantwortung der Frage, ob eine Boolesche Funktion eine Tautologie oder eine Kontradiktion ist.

Das Programm ist ebenfalls in der Lage zu überprüfen, ob ein logischer Ausdruck erfüllbar ist. Obwohl der Begriff der Erfüllbarkeit zu den Grundlagen der Aussagenlogik gehört und zunächst auch leicht nachzuvollziehen ist, ist dessen Bedeutung für konkrete Anwendungen im Unterricht sehr schwer zu vermitteln. Die Komplexitätstheorie, in welcher die Erfüllbarkeit unter anderem angewandt wird, kann bestenfalls im vertiefenden Wahlpflichtfach angeschnitten werden. Dabei könnte etwa die Erfüllbarkeit aussagenlogischer Funktionen als Beispiel für ein NP-Problem herangezogen werden. Da die Iteration aller möglichen Wertebelegungen die einzige Möglichkeit ist, die Erfüllbarkeit einer Funktion zu überprüfen, ermittelt auch das Programm die Erfüllbarkeit auf diese Weise.

Mit Hilfe von Gatterschaltungen ist es auch möglich, einfache Rechengesetze, wie das Assoziativ- oder Distributivgesetz zu veranschaulichen. So ist der Ausdruck $a \wedge (b \wedge c)$ gemäß dem Assoziativgesetz äquivalent zu $(a \wedge b) \wedge c$. Der Ausdruck $a \wedge (b \vee c)$ ist äquivalent zu $(a \wedge b) \vee (a \wedge c)$ aber nicht zu $a \wedge (b \wedge c)$. Die Schülerinnen und Schüler können selbstständig jeweils einen Ausdruck als Gatterschaltung konstruieren und den anderen als Funktion eingeben. Das Programm überprüft dann die Äquivalenz der beiden Ausdrücke. Auf diese Weise können sogar die De-Morgan‘schen Gesetze im Unterricht erarbeitet werden. Die Schülerinnen und Schüler können selbstständig durch trial and error und/oder logisches Denken herausfinden, welcher Ausdruck etwa äquivalent zu $\neg(\mathrm{a} \wedge b)$ ist. Grenzen sind mit dem Programm insofern gesetzt, als dass es nicht in der Lage ist, zwei Logikgatter auf Äquivalenz zu prüfen. Bei der Erarbeitung von Äquivalenzen muss daher immer ein Teil als Funktion und einer als Gatterschaltung eingegeben werden. Die Schaltung kann danach wieder als Funktion angezeigt werden. Insofern sollte diese Einschränkung nicht lernhinderlich sein, sondern im Gegenteil die Vorstellung von der Äquivalenz der beiden Darstellungsformen fördern.

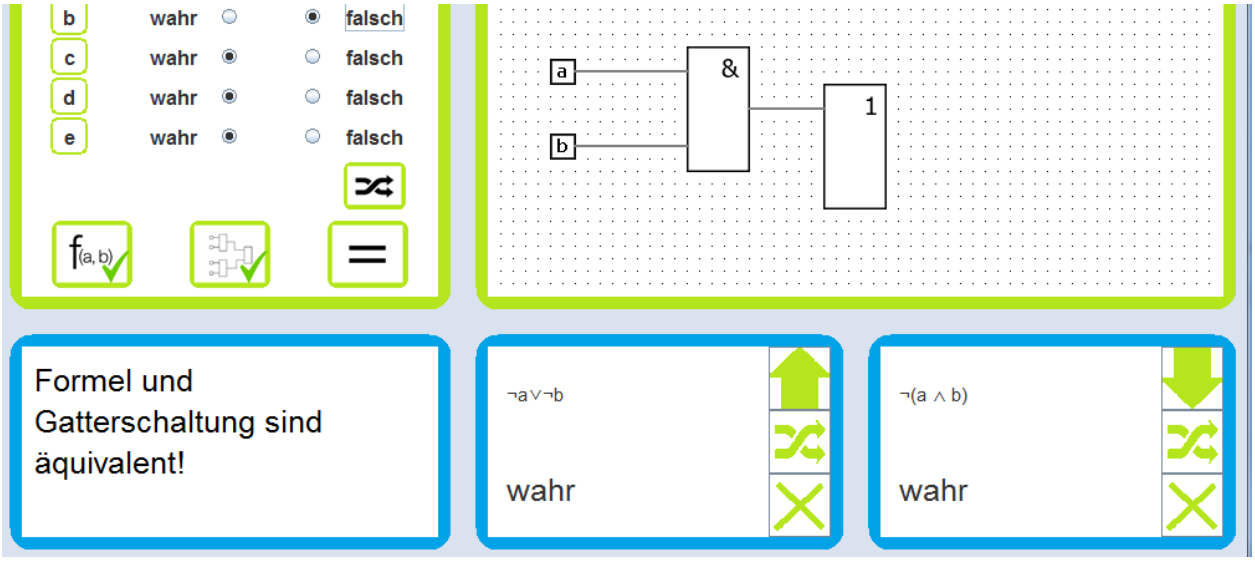


Abb.8: Zeigen einer De-Morgan'schen Regel mit Hilfe des Programms.

Im Lehrplan des Wahlpflichtgegenstandes Informatik für die sechste bis achte Klasse sind *„Aufbau und Funktionsweise von Netzwerken"* sowie *„Erweiterung der theoretischen und technischen Grundlagen der Informatik"*[5] vorgesehen. In diesem Zusammenhang kann im Unterricht die Zuverlässigkeit von Netzwerken thematisiert werden. Für Netzwerke, die aus mehreren Komponenten bestehen, die in Hinsicht auf ihre Zuverlässigkeit voneinander unabhängig sind, kann eine Gesamtzuverlässigkeit berechnet werden. Ist die Integrität eines Netzwerks von der gleichzeitigen Funktion zweier Komponenten abhängig, so werden diese als in Serie geschaltet bezeichnet. Ist die Netzwerkintegrität nur von der Funktion zumindest einer der beiden Komponenten abhängig, so spricht man von einer Parallelschaltung der Komponenten. Die Serienschaltung von Komponenten entspricht einer Verknüpfung mit AND-Operatoren, die Parallelschaltung einer mit OR-Operatoren.[6] Entsprechende Schaltungen können mit dem Programm dargestellt werden. Eine mögliche Aufgabenstellung für den Unterricht wäre, die Schülerinnen und Schüler eine Netzwerkschaltung konstruieren zu lassen, die eine bestimmte Gesamtzuverlässigkeit hat, wenn die Ausfallswahrscheinlichkeiten der einzelnen Komponenten gegeben sind.

Auch außerhalb des unmittelbaren Informatikunterrichts finden sich Probleme, die mit Hilfe einer Gatterschaltung verdeutlicht werden können. In der Physik etwa, können

[5] Bundesministerium für Unterricht, Kunst und Kultur (Hg.): Lehrplan für den Wahlpflichtgegenstand Informatik in der AHS Oberstufe. Wien 2004.
[6] Grossman, Schinagl, Cenker, Neuwirth, Ledl: Netzwerkanalyse: Details. http://startistik.csd.univie.ac.at/netz/details.jsp, abgerufen am 30.01.2012.

elektrische Schaltkreise, die auch zu den technischen Grundlagen der Informatik gehören, mittels Gattern modelliert werden. Serien- und Parallelschaltungen von elektrischen Bauteilen können auf dieselbe Weise dargestellt werden, wie oben für Netzwerkkomponenten beschrieben.

8 Entwicklertagebuch

Das vorliegende Entwicklertagebuch soll wichtige Entwicklungsschritte in meinem Arbeitsverlauf dokumentieren. Aufgeführt sind wesentlichen Neuerungen und Änderungen in der Programmstruktur, sowie der jeweils aktuelle Fortschritt in der Programmentwicklung. Nicht aufgeführt sind unzählige kleinere Entwicklungen, Verbesserungen und Erweiterungen, die zusammen einen wesentlichen Teil des Programms ausmachen. Das Tagebuch enthält Strukturen und Elemente, die in der finalen Programmversion nicht mehr zu finden sind, aber seine Entwicklung dokumentieren.

29., 30. April 2011 Erstellung des Pflichtenhefts.

4. Mai 2011 Abnahme des Pflichtenhefts durch den Betreuer.

23., **24. Mai 2011** Recherche über Java2D, Java Graphics. Überlegungen zu Darstellung Boolescher Funktionen als Baumstruktu Überlegungen zum Analysealgorithmus.

8. Juni 2011 Erstellung erster Klassen- und Anwendungsfall-diagramme.

13. Juni 2011 Abnahme der Diagramme durch den Betreuer.

14., 15. November 2011 Serialisierung zum persistenten Speichern von Objekten implementiert.

3., 4. August 2011 Implementierung einer Baumstruktur zur Darstellung und eines Algorithmus‘ zur Zerlegung Boolescher Funktionen.

8.- 11. August 2011 Entwicklung eines iterativen und eines rekursiven Algorithmus‘ zur Zerlegung Boolescher Funktionen in eine Baumstruktur. Hinzufügen von Klassen für unterschiedliche Gatterarten. Verwendung des Factory-Patterns zur Erzeugung der Klassen. Modellierung der

	Gatter als JButtons.
14. August 2011	Migration der bisherigen graphischen Oberfläche von AWT nach Swing, da dieses Double Buffering unterstützt.
17.-20. August 2011	Implementierung eines Algorithmus zum Vergleich Boolescher Funktionen.
21., 22. August 2011	Zeichnen von Gattern mit Hilfe von GridbagLayout implementiert.
28.-31. August 2011	Zeichnen von Gattern mit Hilfe von AbsoluteLayout implementiert. Algorithmus zur automatischen Anordnung von vorgegeben Plätzen für Gatter implementiert. Algorithmus für das Zeichnen von Verbindungslinien mittels Drag and Drop implementiert. Gatter bekommen DockingPoints, an denen die Linien anschließen.
1. September 2011	Operanden als eigene Klasse hinzugefügt.
5., 6. September 2011	Algorithmus zur Konstruktion von Baumstrukturen aus Gatterschaltungen implementiert.
22.-28. September 2011	Verwendung des Proxy Patterns für die Initialisierung von Gattern.
1.-2. November 2011	Grundlegende Neustrukturierung der Zeichenebene: Verzicht auf fixe Plätze für Gatter und Operanden. Stattdessen ein Grid, auf dem Gatter und Operanden frei platziert werden können. Damit auch Verzicht auf Proxys.
3.-5. November 2011	Neue Implementierung eines Algorithmus‘ zum Zeichnen und Verschieben von Verbindungslinien.

7.-9. November 2011	Gescheiterter Versuch, auf DockingPoints zu verzichten. Drag and Drop für Gatter und Operanden. Verbesserungen am Linienalgorithmus.
9., 10. November 2011	Algorithmus zum automatischen Zeichen von Gatterschaltungen implementiert.
11. November 2011	Negationen für Gatter grafisch und formal hinzugefügt.
12., 13. November 2011	Linienalgorithmus wesentlich verbessert. Selektion von Gattern und Verbindungslinien hinzugefügt.
23. November 2011	Export der Gatterschaltung als JPEG hinzugefügt.
24. November 2011	Menüleiste hinzugefügt.
25.-27. November 2011	Projekt auf das DAO Pattern umgestellt.
28. November 2011	XOR, NAND und NOR als weitere Operatoren hinzugefügt.
29. November 2011	QualifyerManager als zentrale Verwaltung von Qualifyern hinzugefügt. Linienalgorithmus fertiggestellt.
30. November 2011	Hinzufügen von Aufgaben. Aufgabenverwaltung übernimmt ExerciseService, notwendige GUI Elemente befinden sich in ExerciseMenuBar.
1. Dezember 2011	Hinzufügen unterschiedlicher Aufgabenarten. Erstellen von Aufgaben möglich.
3. Dezember 2011	Bearbeiten von Aufgaben möglich.
8., 9. Jänner 2012	Umstellen auf Model-View-Controller Pattern. Trennung von GUI und Logik. Aufteilung der Programmlogik auf mehrere Serviceklassen.

29., 30. Jänner 2012	Grundlegende Neustrukturierung der Zeichenebene: Gatter werden nicht mehr als JButtons dargestellt. Gatter und Verbindungslinien werden gemeinsam in einem Graph-Objekt gespeichert.
13.-15. Februar 2012	Finales Grunddesign der GUI: Textfelder für Ein- und Ausgabe Boolescher Funktionen neu platziert. Feld zur Ausgabe von Texten hinzugefügt.
16.-26. Februar 2012	von Icons zur Gestaltung der GUI. Anpassungen an Bildschirmauflösungen.

Abbildungen

Sämtliche Abbildungen in dieser Arbeit wurden von mir selbst erstellt. Die Abbildungen 6 - 9 wurden mit Hilfe des von mir entwickelten Programms erzeugt.

Abbildung 1: *Anwendungsfalldiagramm*.
Abbildung 2: *Umsetzung des MVC Musters im Programm*.
Abbildung 3: *Umsetzung des DAO Musters im Programm*.
Abbildung 4: *Klassendiagramm des Pakets domain.gui*.
Abbildung 5: *Screenshot des Programms*.
Abbildung 6: *Darstellung eines AND Operators durch NAND-Gatter*.
Abbildung 7: *Ein Halbaddierer, aufgebaut aus XOR- und AND-Gattern*.
Abbildung 8: *Zeigen einer De-Morgan'schen Regel mit Hilfe des Programms*.

Literatur

Balzert, Helmut: Lehrbuch der Software-Technik. Band 1: Spektrum-Akademischer Verlag. Heidelberg 2000.

Bundesministerium für Unterricht, Kunst und Kultur (Hg.): Lehrplan für den Wahlpflichtgegenstand Informatik in der AHS Oberstufe. Wien 2004.

Bundesministerium für Unterricht, Kunst und Kultur (Hg.): Lehrplan Informatik für die AHS Oberstufe. Wien 2004.

Free Software Foundation: GNU General Public Licence. Version 3. 29.6.2007. http://www.gnu.org/licenses/gpl-3.0.html, abgerufen am 1.3.2012.

Gerwinski, Peter: GNU General Public Licence. Deutsche Übersetzung der Version 3. 29.6.2007. http://www.gnu.de/documents/gpl.de.html, abgerufen am 1.3.2012

Grossmann, Schinagl, Cenker, Neuwirth, Ledl: Netzwerkanalyse: Details. http://startistik.csd.univie.ac.at/netz/details.jsp, abgerufen am 30.01.2010.

Javadoc

Overview

Package Summary		Page
dao		47
domain		51
domain.exercise		54
domain.gui		68
domain.qualifyers		92
gui		98
gui.container		118
gui.menubar		126
gui.popup		133
service		138
utils		178

Package

dao

Class Summary		Page
ExerciseDAO	DAO for file access.	48
FileDAO	DAO for file access.	49

dao
ExerciseDAO

```
java.lang.Object
  └ dao.ExerciseDAO
```

```
public class ExerciseDAO
extends Object
```

DAO for file access. `Exercise`s are loaded from and saved to files by this DAO.

Constructor Summary	Page
`ExerciseDAO()`	48

Method Summary		Page
`static ArrayList<Exercise>`	`load(File file)` Loads a list of `Exercise`s from a given file.	49
`static void`	`save(File file, ArrayList<Exercise> exercises)` Saves a list of `Exercise`s to a given file.	48

Constructor Detail

ExerciseDAO

```
public ExerciseDAO()
```

Method Detail

save

```
public static void save(File file,
                        ArrayList<Exercise> exercises)
                 throws IllegalArgumentException,
                        Exception
```

Saves a list of `Exercise`s to a given file.

Parameters:

`file` - The file where the `Exercise`s should be stored.

`exercises` - The `Exercise`s to save.

Throws:

`IllegalArgumentException` - If `exercises` is `null` or `file` is `null`.

`Exception` - If any error occurs while writing to the file.

---

load

```
public static ArrayList<Exercise> load(File file)
                                throws IllegalArgumentException,
                                       Exception
```

Loads a list of `Exercise`s from a given file.

Parameters:

`file` - The file from that the data should be loaded.

Returns:

List of loaded `Exercise`s.

Throws:

`IllegalArgumentException` - If `file` is `null`.

`Exception` - If any error occurs while reading the file.

dao
FileDAO

```
java.lang.Object
  └ dao.FileDAO
```

```
public class FileDAO
extends Object
```

DAO for file access. Loads and saves program states (contained by a `SerializationContainer` from or to files. BufferedImages are saved as jpeg-files.

Constructor Summary		Page
`FileDAO()`		50
Method Summary		**Page**
`static void`	**`export`**`(File file, BufferedImage image)` Exports a `BufferedImage` as a jpeg-file.	50
`static SerializationContainer`	**`load`**`(File file)` Loads a program state from a given file.	51
`static void`	**`save`**`(File file, SerializationContainer save)` Saves a `SerializationContainer` to a given file.	50

Constructor Detail

FileDAO

```
public FileDAO()
```

Method Detail

save

```
public static void save(File file,
                        SerializationContainer save)
                 throws IllegalArgumentException,
                        Exception
```

Saves a `SerializationContainer` to a given file.

Parameters:

`file` - The file where the data should be stored.

`save` - The data to save.

Throws:

`IllegalArgumentException` - If the file is null.

`Exception` - If any error occurs while writing the data.

---

export

```
public static void export(File file,
                          BufferedImage image)
                   throws IllegalArgumentException,
                          Exception
```

Exports a `BufferedImage` as a jpeg-file.

Parameters:

`file` - The file where the image should be stored.

`image` - The image to export.

Throws:

`IllegalArgumentException` - If the given file is null.

`Exception` - If any error occurs while saving the image.

---

load

```
public static SerializationContainer load(File file)
                                   throws IllegalArgumentException,
                                          Exception
```

Loads a program state from a given file. The data read is stored in a `SerializationContainer`.

Parameters:

`file` - The file where the data should be loaded from.

Returns:

The loaded `SerializationContainer`.

Throws:

`IllegalArgumentException` - If the file is null.

`Exception` - If any error occurs while reading the file.

Package

domain

Class Summary		Page
Graph	A transfer object containing all data necessary to draw a logic circuit.	52

domain

Graph

```
java.lang.Object
  └ domain.Graph
```

All Implemented Interfaces:

Serializable

```
public class Graph
extends Object
implements Serializable
```

A transfer object containing all data necessary to draw a logic circuit. It consists of `DrawableComponent`s and `GateConnections`.

Constructor Summary	Page
Graph() Constructs a new `Graph`.	52
Graph(Graph g) Constructs a new `Graph` and deep copies the given `Graph`.	53

Method Summary		Page
void	**addComponent**(DrawableComponent component) Adds a `DrawableComponent` to the logic circuit.	54
void	**addConnection**(GateConnection connection) Adds a `GateConnection` to the logic circuit.	54
void	**clear**() Clears the logic circuit.	54
void	**deleteComponent**(DrawableComponent component) Deletes a `DrawableComponent` from the logic circuit.	53
void	**deleteConnection**(ArrayList<GateConnection> connections) Deletes a list of `GateConnection`s from the logic circuit.	53
void	**deleteConnection**(GateConnection connection) Deletes a `GateConnection` from the logic circuit.	53
ArrayList<GateConnection>	**getConnections**() Returns the `GateConnection`s of the logic circuit.	53
ArrayList<DrawableComponent>	**getDrawables**() Returns the `DrawableComponent`s of the logic circuit.	53
DrawableComponent	**getRootGate**() Returns the root of the logic circuit.	53

Constructor Detail

Graph

```
public Graph()
```

Constructs a new `Graph`. Lists `drawables` and `connections` are initialized.

Graph

`public` **`Graph`**`(Graph g)`

Constructs a new `Graph` and deep copies the given `Graph`. All `GateConnection`s and `DrawableComponent`s including their `DockingPoint`s are deep copied.

Parameters:

`g` - The `Graph` to copy.

Method Detail

getDrawables

`public ArrayList<DrawableComponent>` **`getDrawables`**`()`

Returns the `DrawableComponent`s of the logic circuit.

Returns:

The `DrawableComponent`s of the logic circuit.

---

getConnections

`public ArrayList<GateConnection>` **`getConnections`**`()`

Returns the `GateConnection`s of the logic circuit.

Returns:

The `GateConnection`s of the logic circuit.

---

getRootGate

`public DrawableComponent` **`getRootGate`**`()`

Returns the root of the logic circuit. Root is a `DrawableComponent` with no connection at the `OutDockingPoint`.

Returns:

The root of the logic circuit.

---

deleteComponent

`public void` **`deleteComponent`**`(DrawableComponent component)`

Deletes a `DrawableComponent` from the logic circuit. All its `GateConnection`s are deleted as well.

Parameters:

`component` - The `DrawableComponent` to delete.

---

deleteConnection

`public void` **`deleteConnection`**`(ArrayList<GateConnection> connections)`

Deletes a list of `GateConnection`s from the logic circuit.

Parameters:

`connections` - The `GateConnection`s to delete.

---

deleteConnection

`public void` **`deleteConnection`**`(GateConnection connection)`

Deletes a `GateConnection` from the logic circuit.

Parameters:

`connection` - The `GateConnection` to delete.

---

addComponent

`public void` **`addComponent`**`(DrawableComponent component)`

Adds a `DrawableComponent` to the logic circuit.

Parameters:

`component` - The component to add.

---

addConnection

`public void` **`addConnection`**`(GateConnection connection)`

Adds a `GateConnection` to the logic circuit.

Parameters:

`connection` - The connection to add.

---

clear

`public void` **`clear`**`()`

Clears the logic circuit. All `DrawableComponent`s and `GateConnection`s are deleted.

Package

domain.exercise

Class Summary		Page
Exercise	This class represents an exercise and hold all its data.	55
FunctionExercise	`Exercise` in that a user has to find Boolean function equal to a given logic circuit.	60
GraphExercise	`Exercise` in that a user has to draw a logic circuit equal to a given Boolean function.	62
IncorrectFunctionExercise	`Exercise` in that a user has to correct a given Boolean function so that it becomes equal to a given logic circuit.	64
IncorrectGraphExercise	`Exercise` in that a user has to correct a given logic circuit so that it becomes equal to a given Boolean function.	66

domain.exercise
Exercise

```
java.lang.Object
  └ domain.exercise.Exercise
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

FunctionExercise, GraphExercise

```
public abstract class Exercise
extends Object
implements Serializable
```

This class represents an exercise and hold all its data.

Constructor Summary	Page
`Exercise(int attempts, int points)` Constructs a new `Exercise`.	56

Method Summary		Page
`int`	`getAttemptsAllowed()` Returns the number of attempts a user is allowed to make on this exercise.	56
`int`	`getAttemptsNeeded()` Returns the number of attempts a user already made on this exercise.	56
`abstract String`	`getInstruction()` Returns a message containing instructions what to do in this exercise.	58
`int`	`getPointsReached()` Returns the number of points a user reached when he solved the exercise.	57
`int`	`getPointsWorth()` Returns the number of points this exercise is worth when solved correctly at the first attempt.	57
`void`	`giveUp()` Sets `givenUp` to `True`.	58
`boolean`	`isAccomplished()` Returns `True` if a user has already accomplished this exercise.	58
`boolean`	`isGivenUp()` Returns `True` if the user gave up on this exercise.	58

boolean	**isTried**() Defines if this exercise was already tried by a user.	57
void	**setAccomplished**(boolean accomplished) Sets accomplished to a given boolean value.	58
void	**setAttemptsAllowed**(int attemptsAllowed) Sets the number of attempts a user is allowed to make on this exercise.	56
void	**setAttemptsNeeded**(int attemptsNeeded) Sets the number of attempts a user already made on this exercise.	57
void	**setPointsReached**(int pointsReached) Sets the number of points a user reached when he solved the exercise.	57
void	**setPointsWorth**(int pointsWorth) Sets the number of points this exercise is worth when solved correctly at the first attempt.	57
void	**setTried**(boolean tried) Sets tried to a given boolean value.	57

Constructor Detail

Exercise

```
public Exercise(int attempts,
                int points)
```

Constructs a new Exercise.

Parameters:

attempts - Number of attempts a user is allowed to make on this exercise.

points - Number of points this exercise is worth when solved at the first attempts.

Method Detail

getAttemptsAllowed

```
public int getAttemptsAllowed()
```

Returns the number of attempts a user is allowed to make on this exercise.

Returns:

The number of attempts a user is allowed to make on this exercise.

---

setAttemptsAllowed

```
public void setAttemptsAllowed(int attemptsAllowed)
```

Sets the number of attempts a user is allowed to make on this exercise.

Parameters:

attemptsAllowed - The number of attempts a user is allowed to make on this exercise.

---

getAttemptsNeeded

```
public int getAttemptsNeeded()
```

Returns the number of attempts a user already made on this exercise.

Returns:

The number of attempts a user already made on this exercise.

---

setAttemptsNeeded

`public void` **`setAttemptsNeeded`**`(int attemptsNeeded)`

Sets the number of attempts a user already made on this exercise.

Parameters:

`attemptsNeeded` - Sets the number of attempts a user already made on this exercise.

---

getPointsWorth

`public int` **`getPointsWorth`**`()`

Returns the number of points this exercise is worth when solved correctly at the first attempt.

Returns:

The number of points this exercise is worth when solved correctly at the first attempt.

---

setPointsWorth

`public void` **`setPointsWorth`**`(int pointsWorth)`

Sets the number of points this exercise is worth when solved correctly at the first attempt.

Parameters:

`pointsWorth` - The number of points this exercise is worth when solved correctly at the first attempt.

---

getPointsReached

`public int` **`getPointsReached`**`()`

Returns the number of points a user reached when he solved the exercise.

Returns:

The number of points a user reached when he solved the exercise.

---

setPointsReached

`public void` **`setPointsReached`**`(int pointsReached)`

Sets the number of points a user reached when he solved the exercise.

Parameters:

`pointsReached` - The number of points a user reached when he solved the exercise.

---

isTried

`public boolean` **`isTried`**`()`

Defines if this exercise was already tried by a user.

Returns:

`True` if a user has already made an attempt on this exercise. `False` otherwise.

---

setTried

`public void` **`setTried`**`(boolean tried)`

Sets `tried` to a given boolean value.

Parameters:

`tried` - `True` if a user has already made an attempt on this exercise. `False` otherwise.

---

isAccomplished

`public boolean` **`isAccomplished`**`()`

Returns `True` if a user has already accomplished this exercise.

Returns:

`True` if a user has already accomplished this exercise. `False` otherwise.

---

setAccomplished

`public void` **`setAccomplished`**`(boolean accomplished)`

Sets `accomplished` to a given boolean value.

Parameters:

`accomplished` - `True` if a user has already accomplished this exercise. `False` otherwise.

---

giveUp

`public void` **`giveUp`**`()`

Sets `givenUp` to `True`.

---

isGivenUp

`public boolean` **`isGivenUp`**`()`

Returns `True` if the user gave up on this exercise.

Returns:

`True` if the user gave up on this exercise. `False` otherwise.

---

getInstruction

`public abstract String` **`getInstruction`**`()`

Returns a message containing instructions what to do in this exercise.

Returns:

A message containing instructions what to do in this exercise.

domain.exercise
ExerciseType

```
java.lang.Object
  └ java.lang.Enum<ExerciseType>
      └ domain.exercise.ExerciseType
```

All Implemented Interfaces:

Comparable<ExerciseType>, Serializable

```
public enum ExerciseType
extends Enum<ExerciseType>
```

The types of Exercises.

Enum Constant Summary	Page
DrawGates Find a logic circuit equal to a given Boolean function.	59
FindFunction Find a Boolean function equal to a given logic circuit.	59
IncorFunction Change a given Boolean function so that it becomes equal to a given logic circuit.	59
IncorGraph Change a given logic circuit so that is becomes equal to a given Boolean function.	59

Method Summary		Page
static ExerciseType	**valueOf**(String name)	60
static ExerciseType[]	**values**()	60

Enum Constant Detail

FindFunction

```
public static final ExerciseType FindFunction
```

Find a Boolean function equal to a given logic circuit.

IncorFunction

```
public static final ExerciseType IncorFunction
```

Change a given Boolean function so that it becomes equal to a given logic circuit.

IncorGraph

```
public static final ExerciseType IncorGraph
```

Change a given logic circuit so that is becomes equal to a given Boolean function.

DrawGates

```
public static final ExerciseType DrawGates
```

Find a logic circuit equal to a given Boolean function.

Method Detail

values

```
public static ExerciseType[] values()
```

---

valueOf

```
public static ExerciseType valueOf(String name)
```

domain.exercise
FunctionExercise

```
java.lang.Object
  └ domain.exercise.Exercise
      └ domain.exercise.FunctionExercise
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

IncorrectFunctionExercise

```
public class FunctionExercise
extends Exercise
```

`Exercise` in that a user has to find Boolean function equal to a given logic circuit. The logic circuit and the found solution are stored in this class.

Field Summary		Page
static String	**INSTRUCTION** The instruction what to do in this exercise: "Finde die Funktion zur Gatterschaltung!".	61

Constructor Summary	Page
FunctionExercise(int attempts, int points, Graph graph) Constructs a new `FunctionExercise`.	61

Method Summary		Page
Graph	**getGraph**() Returns the logic circuit of this exercise.	62
String	**getInstruction**() Returns the instruction for this exercise.	62
String	**getSolution**() Returns a users solution of this exercise.	62
void	**setGraph**(Graph graph) Sets the logic circuit of this exercise.	62
void	**setSolution**(String function) Sets a users solution of this exercise.	62

Field Detail

INSTRUCTION

```
public static final String INSTRUCTION
```

The instruction what to do in this exercise: "Finde die Funktion zur Gatterschaltung!".

Constructor Detail

FunctionExercise

```
public FunctionExercise(int attempts,
                        int points,
                        Graph graph)
```

Constructs a new `FunctionExercise`.

Parameters:

`attempts` - Number of attempts a user is allowed to make on this exercise.

`points` - Number of points this exercise is worth when solved at the first attempts.

`graph` - The logic circuit to that an equal Boolean function has to be found by the user.

Method Detail

getGraph

`public Graph getGraph()`

Returns the logic circuit of this exercise.

Returns:

The logic circuit of this exercise.

---

setGraph

`public void setGraph(Graph graph)`

Sets the logic circuit of this exercise.

Parameters:

`graph` - The logic circuit of this exercise.

---

setSolution

`public void setSolution(String function)`

Sets a users solution of this exercise.

Parameters:

`function` - A users solution of this exercise.

---

getSolution

`public String getSolution()`

Returns a users solution of this exercise.

Returns:

A users solution of this exercise.

---

getInstruction

`public String getInstruction()`

Returns the instruction for this exercise.

Overrides:

`getInstruction` in class `Exercise`

Returns:

A message containing instructions what to do in this exercise.

domain.exercise
GraphExercise

```
java.lang.Object
  └ domain.exercise.Exercise
      └ domain.exercise.GraphExercise
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

IncorrectGraphExercise

```
public class GraphExercise
extends Exercise
```

`Exercise` in that a user has to draw a logic circuit equal to a given Boolean function. The Boolean function and the users solution are stored in this class.

Field Summary		Page
static String	**INSTRUCTION** The instruction what to do in this exercise: "Zeichne die äquivalente Gatterschaltung!".	63

Constructor Summary	Page
GraphExercise(int attempts, int points, String function) Constructs a new GraphExercise.	63

Method Summary		Page
String	**getFunction**() Returns the Boolean function to that a user has to find a logic circuit.	64
String	**getInstruction**() Returns a message containing instructions what to do in this exercise.	64
Graph	**getSolution**() Returns a logic circuit that was drawn by a user as a solution for this exercise.	64
void	**setFunction**(String function) Sets the Boolean function to that a user has to find a logic circuit.	64
void	**setSolution**(Graph graph) Sets a logical circuit as a solution that was found by a user.	64

Field Detail

INSTRUCTION

```
public static final String INSTRUCTION
```

The instruction what to do in this exercise: "Zeichne die äquivalente Gatterschaltung!".

Constructor Detail

GraphExercise

```
public GraphExercise(int attempts,
                     int points,
                     String function)
```

Constructs a new GraphExercise.

Parameters:

`attempts` - Number of attempts a user is allowed to make on this exercise.

`points` - Number of points this exercise is worth when solved at the first attempts.

`function` - The Boolean function to that a user has to find an equal logic circuit.

Method Detail

getFunction

`public String` **`getFunction`**`()`

Returns the Boolean function to that a user has to find a logic circuit.

Returns:

The Boolean function to that a user has to find a logic circuit.

---

setFunction

`public void` **`setFunction`**`(String function)`

Sets the Boolean function to that a user has to find a logic circuit.

Parameters:

`function` - A Boolean function to that a user has to find a logic circuit.

---

setSolution

`public void` **`setSolution`**`(Graph graph)`

Sets a logical circuit as a solution that was found by a user.

Parameters:

`graph` - a logical circuit as a solution that was found by a user.

---

getSolution

`public Graph` **`getSolution`**`()`

Returns a logic circuit that was drawn by a user as a solution for this exercise.

Returns:

A logic circuit that was drawn by a user as a solution for this exercise.

---

getInstruction

`public String` **`getInstruction`**`()`

Description copied from class: `Exercise`

Returns a message containing instructions what to do in this exercise.

Overrides:

`getInstruction` in class `Exercise`

Returns:

A message containing instructions what to do in this exercise.

domain.exercise
IncorrectFunctionExercise

```
java.lang.Object
  └ domain.exercise.Exercise
      └ domain.exercise.FunctionExercise
          └ domain.exercise.IncorrectFunctionExercise
```

All Implemented Interfaces:

Serializable

```
public class IncorrectFunctionExercise
extends FunctionExercise
```

`Exercise` in that a user has to correct a given Boolean function so that it becomes equal to a given logic circuit. The logic circuit, the incorrect function and the found solution are stored in this class.

Field Summary		Page
static String	**INSTRUCTION** The instruction what to do in this exercise: "Finde den Fehler in der Funktion!".	65

Constructor Summary	Page
IncorrectFunctionExercise(int attempts, int points, String function, Graph graph) Constructs a new `IncorrectFunctionExercise`.	65

Method Summary		Page
String	**getFunction**() Returns the Boolean function that has to be changed so that it becomes equal to the logic circuit.	66
String	**getInstruction**() Returns the instruction for this exercise.	66
void	**setFunction**(String function) Sets the Boolean function that has to be changed so that it becomes equal to the logic circuit.	66

Field Detail

INSTRUCTION

```
public static final String INSTRUCTION
```

The instruction what to do in this exercise: "Finde den Fehler in der Funktion!".

Constructor Detail

IncorrectFunctionExercise

```
public IncorrectFunctionExercise(int attempts,
                                 int points,
                                 String function,
                                 Graph graph)
```

Constructs a new `IncorrectFunctionExercise`.

Parameters:

`attempts` - Number of attempts a user is allowed to make on this exercise.

`points` - Number of points this exercise is worth when solved at the first attempts.

`function` - Boolean function that is not equal to the logic circuit.

`graph` - The logic circuit to that an equal Boolean function has to be found by the user.

Method Detail

getFunction

`public String` **`getFunction`**`()`

Returns the Boolean function that has to be changed so that it becomes equal to the logic circuit.

Returns:

The Boolean function that has to be changed so that it becomes equal to the logic circuit.

---

setFunction

`public void` **`setFunction`**`(String function)`

Sets the Boolean function that has to be changed so that it becomes equal to the logic circuit.

Parameters:

`function` - The Boolean function that has to be changed so that it becomes equal to the logic circuit.

---

getInstruction

`public String` **`getInstruction`**`()`

Description copied from class: `FunctionExercise`

Returns the instruction for this exercise.

Overrides:

`getInstruction` in class `FunctionExercise`

Returns:

A message containing instructions what to do in this exercise.

domain.exercise
IncorrectGraphExercise

```
java.lang.Object
  └ domain.exercise.Exercise
      └ domain.exercise.GraphExercise
          └ domain.exercise.IncorrectGraphExercise
```

All Implemented Interfaces:

Serializable

```
public class IncorrectGraphExercise
extends GraphExercise
```

Exercise in that a user has to correct a given logic circuit so that it becomes equal to a given Boolean function. The Boolean function, the incorrect logic circuit and the found solution are stored in this class.

Field Summary		Page
static String	**INSTRUCTION** The instruction what to do in this exercise: "Finde den Fehler in der Gatterschaltung!".	67

Constructor Summary	Page
IncorrectGraphExercise(int attempts, int points, String function, Graph graph) Constructs a new IncorrectGraphExercise.	67

Method Summary		Page
Graph	**getGraph**() Returns an incorrect logic circuit that the user has to change so that it becomes equal to the given Boolean function.	68
String	**getInstruction**() Returns a message containing instructions what to do in this exercise.	68
void	**setGraph**(Graph graph) Sets an incorrect logic circuit that the user has to change so that it becomes equal to the given Boolean function.	68

Field Detail

INSTRUCTION

```
public static final String INSTRUCTION
```

The instruction what to do in this exercise: "Finde den Fehler in der Gatterschaltung!".

Constructor Detail

IncorrectGraphExercise

```
public IncorrectGraphExercise(int attempts,
                              int points,
                              String function,
                              Graph graph)
```

Constructs a new IncorrectGraphExercise.

Parameters:

attempts - Number of attempts a user is allowed to make on this exercise.

points - Number of points this exercise is worth when solved at the first attempts.

function - The Boolean function to that a user has to find an equal logic circuit.

graph - A logic circuit that is not equal to the Boolean function.

Method Detail

getGraph

`public Graph getGraph()`

Returns an incorrect logic circuit that the user has to change so that it becomes equal to the given Boolean function.

Returns:

An incorrect logic circuit that the user has to change so that it becomes equal to the given Boolean function.

---

setGraph

`public void setGraph(Graph graph)`

Sets an incorrect logic circuit that the user has to change so that it becomes equal to the given Boolean function.

Parameters:

`graph` - An incorrect logic circuit that the user has to change so that it becomes equal to the given Boolean function.

---

getInstruction

`public String getInstruction()`

Description copied from class: Exercise

Returns a message containing instructions what to do in this exercise.

Overrides:

`getInstruction` in class `GraphExercise`

Returns:

A message containing instructions what to do in this exercise.

Package

domain.gui

Class Summary		Page
DockingPoint	DockingPoints represent input or output areas of DrawableComponents and work as "hooks" for GateConnections.	69
DrawableComponent	DrawableComponent is the super type of all elements of a logic circuit that are drawn in DrawingArea apart from GateConnections.	74
DrawablesFactory	Factory for DrawableComponents.	77
Gate	Element of a logic circuit representing operators including the NOT-sign.	78
GateConnection	Represents a line connecting two components of a logic circuit.	80
GateNot	Represents a NOT sign in a logic circuit.	83
GateOperator	Represents an operator in a logic circuit.	85
Node	An area attached to a GateConnection that can be used to edit it.	87
Operand	Represents an operand in a logic circuit.	90

Enum Summary		Page
DockType	Defines the types of DockingPoints.	72

domain.gui
DockingPoint

```
java.lang.Object
  └ domain.gui.DockingPoint
```

All Implemented Interfaces:

Serializable

```
public class DockingPoint
extends Object
implements Serializable
```

DockingPoints represent input or output areas of DrawableComponents and work as "hooks" for GateConnections. They can only be established between 2 DockingPoints so that each DrawableComponent has one, two or three of them, depending on the components type. The exact position where a GateConnection starts or ends is calculated in GridService using a DockingPoint. To mark their position on a DrawableComponent each DockingPoint is associated with a DockType.

Field Summary		Page
int	**height** The height of this DockingPoint.	70
int	**width** The width of this DockingPoint.	70
int	**x** The x coordinate of this DockingPoint.	70
int	**y** The y coordinate of this DockingPoint.	70

Constructor Summary	Page
DockingPoint(DockType type, DrawableComponent parent) Constructs a new DockingPoint.	71

Method Summary		Page
void	**addConnection**(GateConnection con) Adds a GateConnection that hooks into this DockingPoint.	71

void	**changeNegation**() Changes the negation of this DockingPoint.	72
boolean	**contains**(Point point) Returns True if this DockingPoint contains a given point or False otherwise.	72
Rectangle	**getBounds**() Returns the bounds of this DockingPoint.	71
ArrayList<GateConnection>	**getConnections**() Returns all GateConnections that are hooked into this DockingPoint.	72
DrawableComponent	**getParent**() Returns the parent DrawableComponent that this dock belongs to.	71
DockType	**getType**() Returns the type of this DockingPoint specifying its position on the parent DrawableComponent.	71
boolean	**isNegated**() Returns True if this DockingPoint is negated.	72
void	**removeConnection**(GateConnection connection) Removes a GateConnection that is hooked into this DockingPoint.	72
void	**setBounds**(Rectangle r) Sets the bounds of this DockingPoint.	71
void	**setNegated**(boolean neg) Sets the negation of this DockingPoint	72

Field Detail

height

public int **height**

The height of this DockingPoint.

width

public int **width**

The width of this DockingPoint.

x

public int **x**

The x coordinate of this DockingPoint.

y

public int **y**

The y coordinate of this DockingPoint.

Constructor Detail

DockingPoint

```
public DockingPoint(DockType type,
                    DrawableComponent parent)
```

Constructs a new `DockingPoint`.

Parameters:

`type` - The `DockType` marking the position on an `DrawableComponent`.

`parent` - The parent `DrawableComponent` that this point belongs to.

Method Detail

getBounds

```
public Rectangle getBounds()
```

Returns the bounds of this `DockingPoint`.

Returns:

The bounds of this `DockingPoint` as a Rectangle.

---

setBounds

```
public void setBounds(Rectangle r)
```

Sets the bounds of this `DockingPoint`.

Parameters:

`r` - The bounds of this `DockingPoint` as a Rectangle.

---

getParent

```
public DrawableComponent getParent()
```

Returns the parent `DrawableComponent` that this dock belongs to.

Returns:

The parent `DrawableComponent` that this dock belongs to.

---

getType

```
public DockType getType()
```

Returns the type of this `DockingPoint` specifying its position on the parent `DrawableComponent`.

Returns:

The type of this `DockingPoint` specifying its position on the parent `DrawableComponent`.

---

addConnection

```
public void addConnection(GateConnection con)
```

Adds a `GateConnection` that hooks into this `DockingPoint`. If this `DockingPoint`s type is not `Out`, all previous connections are deleted.

Parameters:

`con` - The `GateConnection` that hooks into this `DockingPoint`.

---

getConnections

`public ArrayList<GateConnection> getConnections()`

Returns all `GateConnection`s that are hooked into this `DockingPoint`.

Returns:

All `GateConnection`s that are hooked into this `DockingPoint`.

---

isNegated

`public boolean isNegated()`

Returns `True` if this `DockingPoint` is negated. `False` otherwise.

Returns:

`True` if this `DockingPoint` is negated. `False` otherwise.

---

setNegated

`public void setNegated(boolean neg)`

Sets the negation of this `DockingPoint`

Parameters:

`neg` - `True` if this `DockingPoint` should be negated. `False` otherwise.

---

changeNegation

`public void changeNegation()`

Changes the negation of this `DockingPoint`. If its negation was `True` it becomes `False` now. If it was `False` it becomes `True`.

---

removeConnection

`public void removeConnection(GateConnection connection)`

Removes a `GateConnection` that is hooked into this `DockingPoint`.

Parameters:

`connection` - The `GateConnection` that should be removed from this `DockingPoint`.

---

contains

`public boolean contains(Point point)`

Returns `True` if this `DockingPoint` contains a given point or `False` otherwise.

Parameters:

`point` - A point thats position is checked and compared to this `DockingPoint`s position.

Returns:

`True` if this `DockingPoint` contains a given point. `False` otherwise.

domain.gui
DockType

```
java.lang.Object
  └java.lang.Enum<DockType>
      └domain.gui.DockType
```

All Implemented Interfaces:

Comparable<DockType>, Serializable

```
public enum DockType
extends Enum<DockType>
```

Defines the types of `DockingPoint`s. A type specifies a position on a `DrawableComponent` and if a `DockingPoint` represents an input or an output.

Enum Constant Summary	Page
In1 The first (upper) input of a `DrawableComponent`.	73
In2 The second (lower) input of a `DrawableComponent`.	73
Out The output of a `DrawableComponent`.	73

Method Summary		Page
static DockType	**valueOf**(String name)	73
static DockType[]	**values**()	73

Enum Constant Detail

In1

```
public static final DockType In1
```

The first (upper) input of a `DrawableComponent`. Only operator gates have an In1-`DockingPoint`.

In2

```
public static final DockType In2
```

The second (lower) input of a `DrawableComponent`. Only operator gates apart from NOT have an In2-`DockingPoint`.

Out

```
public static final DockType Out
```

The output of a `DrawableComponent`. Each `DrawableComponent` has an Out-`DockingPoint`.

Method Detail

values

```
public static DockType[] values()
```

---

valueOf

```
public static DockType valueOf(String name)
```

domain.gui
DrawableComponent

```
java.lang.Object
  └ domain.gui.DrawableComponent
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

Gate, Operand

```
public abstract class DrawableComponent
extends Object
implements Serializable
```

DrawableComponent is the super type of all elements of a logic circuit that are drawn in DrawingArea apart from GateConnections. A DrawableComponent can be either an operand or and operator. DrawableComponents provide DockingPoints for GateConnections to hook up.

Field Summary		Page
int	**height** The x coordinate of the DrawableComponent.	75
int	**width** The x coordinate of the DrawableComponent.	75
int	**x** The x coordinate of the DrawableComponent.	75
int	**y** The x coordinate of the DrawableComponent.	75

Constructor Summary	Page
DrawableComponent(BooleanQualifyer qualifyer) Constructs a new DrawableComponent at the position (0, 0).	75
DrawableComponent(BooleanQualifyer qualifyer, int x, int y, int width, int height) Constructs a new DrawableComponent at the position (x, y).	75

Method Summary		Page
abstract DrawableComponent	**copy**() Deep copy method for DrawableComponents.	76
HashMap<DockType,DockingPoint>	**getDocks**() Returns all DockingPoints this DrawableComponent provides.	77
BooleanQualifyer	**getQualifyer**() Returns the Qualifyer of this DrawableComponent.	76
boolean	**hasOutgoingConnection**() Returns True if a GateConnection is hooked into the Out-DockingPoint of the DrawableComponent.	76
protected void	**initializeComponent**() Initializes as many DockingPoints as this DrawableComponent needs and stores them in docks.	76
void	**setBounds**(int x, int y, int width, int height) Sets the bounds of the DrawableComponent.	76
void	**setQualifyer**(BooleanQualifyer q) Sets the Qualifyer of this DrawableComponent.	76

Field Detail

height

`public int` **`height`**

The x coordinate of the `DrawableComponent`.

---

width

`public int` **`width`**

The x coordinate of the `DrawableComponent`.

---

x

`public int` **`x`**

The x coordinate of the `DrawableComponent`.

---

y

`public int` **`y`**

The x coordinate of the `DrawableComponent`.

Constructor Detail

DrawableComponent

```
public DrawableComponent(BooleanQualifyer qualifyer,
                         int x,
                         int y,
                         int width,
                         int height)
```

Constructs a new `DrawableComponent` at the position (x, y).

Parameters:

`qualifyer` - Defines the type of the `DrawableComponent`.

`x` - The x coordinate of the `DrawableComponent`.

`y` - The y coordinate of the `DrawableComponent`.

`width` - The width of the `DrawableComponent`.

`height` - The height of the `DrawableComponent`.

DrawableComponent

```
public DrawableComponent(BooleanQualifyer qualifyer)
```

Constructs a new `DrawableComponent` at the position (0, 0).

Parameters:

`qualifyer` - Defines the type of the `DrawableComponent`.

Method Detail

initializeComponent

```
protected void initializeComponent()
```

Initializes as many DockingPoints as this DrawableComponent needs and stores them in docks. DockingPoints are placed on the component according to their and the components types.

---

copy

```
public abstract DrawableComponent copy()
```

Deep copy method for DrawableComponents. The new component has the same coordinates and the same number of DockingPoints. Hooked GateConnections won't be copied.

Returns:

A deep copy of the DrawableComponent.

---

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

Sets the bounds of the DrawableComponent.

Parameters:

x - The new x coordinate of the DrawableComponent.

y - The new y coordinate of the DrawableComponent.

width - The new width of the DrawableComponent.

height - The new height of the DrawableComponent.

---

hasOutgoingConnection

```
public boolean hasOutgoingConnection()
```

Returns True if a GateConnection is hooked into the Out-DockingPoint of the DrawableComponent.

Returns:

True if a GateConnection is hooked into the Out-DockingPoint of the DrawableComponent. False otherwise.

---

setQualifyer

```
public void setQualifyer(BooleanQualifyer q)
```

Sets the Qualifyer of this DrawableComponent.

Parameters:

q - The new Qualifyer of this DrawableComponent.

---

getQualifyer

```
public BooleanQualifyer getQualifyer()
```

Returns the Qualifyer of this DrawableComponent.

Returns:

The Qualifyer of this DrawableComponent.

---

getDocks

`public HashMap<DockType,DockingPoint> getDocks()`

Returns all `DockingPoint`s this `DrawableComponent` provides.

Returns:

All `DockingPoint`s this `DrawableComponent` provides.

domain.gui
DrawablesFactory

```
java.lang.Object
  └ domain.gui.DrawablesFactory
```

```
public class DrawablesFactory
extends Object
```

Factory for `DrawableComponent`s.

Method Summary		Page
static DrawableComponent	**newComponent**(BooleanQualifyer qualifyer) Creates a new `DrawableComponent`.	78
static DrawableComponent	**newDrawable**(DrawableComponent dc) Creates a new `DrawableComponent` that is of the same type as a given one.	78

Method Detail

newComponent

```
public static DrawableComponent newComponent(BooleanQualifyer qualifyer)
```

Creates a new `DrawableComponent`.

Parameters:

`qualifyer` - The `Qualifyer` defining the new element.

Returns:

A new `DrawableComponent`.

---

newDrawable

```
public static DrawableComponent newDrawable(DrawableComponent dc)
```

Creates a new `DrawableComponent` that is of the same type as a given one.

Parameters:

`dc` - The `DrawableComponent` of thats type the new `DrawableComponent` should be.

Returns:

A new `DrawableComponent` that is of the same type as a given one.

domain.gui
Gate

```
java.lang.Object
  └ domain.gui.DrawableComponent
      └ domain.gui.Gate
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

GateNot, GateOperator

```
public abstract class Gate
extends DrawableComponent
```

Element of a logic circuit representing operators including the NOT-sign.

Fields inherited from class domain.gui.DrawableComponent	
height, width, x, y	

Constructor Summary	Page
Gate(QualifyerGate gq) Constructs a new Gate.	79
Gate(QualifyerGate gq, int x, int y, int width, int height) Constructs a new Gate at position (x, y).	79

Method Summary		Page
protected void	**initializeComponent**() Initializes as many DockingPoints as this DrawableComponent needs and stores them in docks.	80

Constructor Detail

Gate

```
public Gate(QualifyerGate gq)
```

Constructs a new Gate.

Parameters:

gq - The Qualifyer defining the type of the element.

Gate

```
public Gate(QualifyerGate gq,
            int x,
            int y,
            int width,
            int height)
```

Constructs a new Gate at position (x, y).

Parameters:

gq - The Qualifyer defining the type of the element.

x - The x coordinate of the Gate.

y - The y coordinate of the Gate.

width - The width of the Gate.

height - The height of the Gate.

Method Detail

initializeComponent

`protected void` **`initializeComponent`**`()`

Description copied from class: **`DrawableComponent`**

Initializes as many `DockingPoint`s as this `DrawableComponent` needs and stores them in `docks`. `DockingPoint`s are placed on the component according to their and the components types.

Overrides:

`initializeComponent` in class `DrawableComponent`

domain.gui
GateConnection

```
java.lang.Object
  └ domain.gui.GateConnection
```

All Implemented Interfaces:

Serializable

```
public class GateConnection
extends Object
implements Serializable
```

Represents a line connecting two components of a logic circuit. A `GateConnection` hooks into a `DockingPoint` that is provided by the component for this purpose. The connection consists of a number of single horizontal and vertical lines.

Constructor Summary	Page
GateConnection(DockingPoint start) Constructs a new `GateConnection`, starting at the given `DockingPoint`	81

Method Summary		Page
boolean	**containsPoint**(Point p) Returns `True` if this `GateConnection` contains a given point.	82
DockingPoint	**getEnd**() Returns the `DockingPoint` at that this connection ends.	82
ArrayList<Line2D.Double>	**getLines**() Returns all singular lines this `GateConnection` consists of.	82
Node	**getNewNode**(Point p, int width, int height) Creates a new `Node` at a line of this connection that is next to at a given point.	83
DockingPoint	**getStart**() Returns the `DockingPoint` at that this connection starts.	82
boolean	**setEnd**(DockingPoint ende) Finalizes the construction of a `GateConnection`.	81
void	**setLines**(ArrayList<Line2D.Double> lines) Sets the lines this `GateConnection` consists of.	82

Constructor Detail

GateConnection

```
public GateConnection(DockingPoint start)
```

Constructs a new `GateConnection`, starting at the given `DockingPoint`

Parameters:

`start` - The `DockingPoint` where this connection should start.

Method Detail

setEnd

```
public boolean setEnd(DockingPoint ende)
```

Finalizes the construction of a `GateConnection`. The connection is now established between the start and the end `DockingPoint`. Start and end docks must not be of the same type. If they are of the

same type, the connection is not finished and `False` is returned. Start `DockingPoint` will be the point that is of the `DockType Out`. End point will be the point that is not of this type.

Parameters:

`ende` - The second point defining the connection. Must not be of the same `DockType` as `start`.

Returns:

`True` if the connection was finalized. `False` otherwise, especially if `end` is of the same `DockType` as `start`.

---

getStart

`public DockingPoint getStart()`

Returns the `DockingPoint` at that this connection starts.

Returns:

The `DockingPoint` at that this connection starts.

---

getEnd

`public DockingPoint getEnd()`

Returns the `DockingPoint` at that this connection ends.

Returns:

The `DockingPoint` at that this connection ends.

---

containsPoint

`public boolean containsPoint(Point p)`

Returns `True` if this `GateConnection` contains a given point. To be contained by this `GateConnection` the point has to be exactly on a line of this connection or close to it.

Parameters:

`p` - The point to check.

Returns:

`True` if this `GateConnection` contains the point. `False` otherwise.

---

getLines

`public ArrayList<Line2D.Double> getLines()`

Returns all singular lines this `GateConnection` consists of.

Returns:

All singular lines this `GateConnection` consists of.

---

setLines

`public void setLines(ArrayList<Line2D.Double> lines)`

Sets the lines this `GateConnection` consists of.

Parameters:

`lines` - The new lines this `GateConnection` consists of.

---

getNewNode

```
public Node getNewNode(Point p,
                       int width,
                       int height)
```

Creates a new `Node` at a line of this connection that is next to at a given point.

Parameters:

`p` - Point next to that the `Node` should be created.

`width` - The width of the `Node`.

`height` - The height of the `Node`.

Returns:

A new Node at a line of this connection that is next to at a given point.

domain.gui
GateNot

```
java.lang.Object
  └ domain.gui.DrawableComponent
      └ domain.gui.Gate
          └ domain.gui.GateNot
```

All Implemented Interfaces:

Serializable

```
public class GateNot
extends Gate
```

Represents a NOT sign in a logic circuit.

Fields inherited from class domain.gui.DrawableComponent
height, width, x, y

Constructor Summary	Page
GateNot(QualifyerGate qg) Constructs a new NOT element.	84
GateNot(QualifyerGate qg, int x, int y, int width, int height) Constructs a new NOT element at a specified position and with a specified dimension.	84

Method Summary		Page
GateNot	**copy**() Deep copy method for DrawableComponents.	85
void	**setBounds**(int x, int y, int width, int height) Sets the bounds of the DrawableComponent.	85

Constructor Detail

GateNot

```
public GateNot(QualifyerGate qg)
```

Constructs a new NOT element.

Parameters:

qg - The Qualifyer defining the type of this component.

GateNot

```
public GateNot(QualifyerGate qg,
               int x,
               int y,
               int width,
               int height)
```

Constructs a new NOT element at a specified position and with a specified dimension.

Parameters:

qg - The Qualifyer defining the type of this component.

x - The x coordinate of this component.

y - The y coordinate of this component.

width - The width of this component.

height - The height of this component.

Method Detail

copy

```
public GateNot copy()
```

Description copied from class: DrawableComponent

Deep copy method for DrawableComponents. The new component has the same coordinates and the same number of DockingPoints. Hooked GateConnections won't be copied.

Overrides:

copy in class DrawableComponent

Returns:

A deep copy of the DrawableComponent.

---

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

Description copied from class: DrawableComponent

Sets the bounds of the DrawableComponent.

Overrides:

setBounds in class DrawableComponent

Parameters:

x - The new x coordinate of the DrawableComponent.

y - The new y coordinate of the DrawableComponent.

width - The new width of the DrawableComponent.

height - The new height of the DrawableComponent.

domain.gui
GateOperator

```
java.lang.Object
  └ domain.gui.DrawableComponent
      └ domain.gui.Gate
          └ domain.gui.GateOperator
```

All Implemented Interfaces:

Serializable

```
public class GateOperator
extends Gate
```

Represents an operator in a logic circuit.

Constructor Summary	Page
GateOperator(QualifyerGate gq) Constructs a new GateOperator.	86
GateOperator(QualifyerGate gq, int x, int y, int width, int height) Constructs a new GateOperator at a specified position and with a specified dimension.	86

Method Summary		Page
GateOperator	**copy**() Deep copy method for DrawableComponents.	87
protected void	**initializeComponent**() Initializes as many DockingPoints as this DrawableComponent needs and stores them in docks.	87
void	**setBounds**(int x, int y, int width, int height) Sets the bounds of the DrawableComponent.	87

Constructor Detail

GateOperator

```
public GateOperator(QualifyerGate gq)
```

Constructs a new GateOperator.

Parameters:

gq - The Qualifyer defining the type of this component.

GateOperator

```
public GateOperator(QualifyerGate gq,
                    int x,
                    int y,
                    int width,
                    int height)
```

Constructs a new GateOperator at a specified position and with a specified dimension.

Parameters:

gq - The Qualifyer defining the type of this component.

x - The x coordinate of this component.

y - The y coordinate of this component.

width - The width of this component.

height - The height of this component.

Method Detail

initializeComponent

```
protected void initializeComponent()
```

Description copied from class: DrawableComponent

Initializes as many DockingPoints as this DrawableComponent needs and stores them in docks. DockingPoints are placed on the component according to their and the components types.

Overrides:

initializeComponent in class Gate

---

copy

```
public GateOperator copy()
```

Description copied from class: DrawableComponent

Deep copy method for DrawableComponents. The new component has the same coordinates and the same number of DockingPoints. Hooked GateConnections won't be copied.

Overrides:

copy in class DrawableComponent

Returns:

A deep copy of the DrawableComponent.

---

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

Description copied from class: DrawableComponent

Sets the bounds of the DrawableComponent.

Overrides:

setBounds in class DrawableComponent

Parameters:

x - The new x coordinate of the DrawableComponent.

y - The new y coordinate of the DrawableComponent.

width - The new width of the DrawableComponent.

height - The new height of the DrawableComponent.

domain.gui
Node

```
java.lang.Object
  └java.awt.geom.Point2D
      └java.awt.Point
          └domain.gui.Node
```

All Implemented Interfaces:

Serializable, Cloneable

```
public class Node
extends Point
```

An area attached to a `GateConnection` that can be used to edit it. A `Node` can be dragged, which drags the line as well. It can be clicked right as well to show a context menu for the particular `GateConnection`.

Constructor Summary	Page
Node(int x, int y, int width, int height) Constructs a new Node at a specified position and with a specified size.	88

Method Summary		Page
boolean	**contains**(int px, int py) Returns True if this Node contains a given point.	89
GateConnection	**getConnection**() Returns the GateConnection that this Node is attached to.	90
int	**getHeight**() Returns the height of this Node.	89
Line2D.Double	**getLine**() Returns the line that this Node is attached to.	89
int	**getWidth**() Returns the width of this Node	89
void	**setConnection**(GateConnection connection) Sets the GateConnection that this Node is attached to.	89
void	**setLine**(Line2D.Double line) Sets the line that this Node is attached to.	89

Constructor Detail

Node

```
public Node(int x,
            int y,
            int width,
            int height)
```

Constructs a new Node at a specified position and with a specified size.

Parameters:

`x` - The x coordinate of the `Node`.

`y` - The y coordinate of the `Node`.

`width` - The width of the `Node`.

`height` - The height of the `Node`.

Method Detail

setLine

```
public void setLine(Line2D.Double line)
```

Sets the line that this `Node` is attached to.

Parameters:

`line` - A line that this `Node` should be attached to.

---

getLine

```
public Line2D.Double getLine()
```

Returns the line that this `Node` is attached to.

Returns:

The line that this `Node` is attached to.

---

contains

```
public boolean contains(int px,
                        int py)
```

Returns `True` if this `Node` contains a given point.

Parameters:

`px` - The x coordinate of the point.

`py` - The y coordinate of the point.

Returns:

`True` if this `Node` contains a given point. `False` otherwise.

---

getWidth

```
public int getWidth()
```

Returns the width of this `Node`

Returns:

The width of this `Node`.

---

getHeight

```
public int getHeight()
```

Returns the height of this `Node`.

Returns:

The height of this `Node`.

---

setConnection

```
public void setConnection(GateConnection connection)
```

Sets the `GateConnection` that this `Node` is attached to.

Parameters:

`connection` - The `GateConnection` that this `Node` is attached to.

---

getConnection

`public GateConnection **getConnection**()`

Returns the `GateConnection` that this `Node` is attached to.

Returns:

The `GateConnection` that this `Node` is attached to.

domain.gui
Operand

```
java.lang.Object
  └ domain.gui.DrawableComponent
      └ domain.gui.Operand
```

All Implemented Interfaces:

Serializable

```
public class Operand
extends DrawableComponent
implements Serializable
```

Represents an operand in a logic circuit.

Constructor Summary	Page
Operand(QualifyerOperand qo) Constructs a new Operand at the position (0, 0).	91
Operand(QualifyerOperand qo, int x, int y, int width, int height) Constructs a new Operand element at a specified position and with a specified dimension.	91

Method Summary		Page
Operand	**copy**() Deep copy method for DrawableComponents.	92
void	**setBounds**(int x, int y, int width, int height) Sets the bounds of the DrawableComponent.	92

Constructor Detail

Operand

```
public Operand(QualifyerOperand qo)
```

Constructs a new Operand at the position (0, 0).

Parameters:

qo - The Qualifyer defining the type of this component.

Operand

```
public Operand(QualifyerOperand qo,
               int x,
               int y,
               int width,
               int height)
```

Constructs a new Operand element at a specified position and with a specified dimension.

Parameters:

qo - The Qualifyer defining the type of this component.

x - The x coordinate of this component.

y - The y coordinate of this component.

width - The width of this component.

height - The height of this component.

Method Detail

copy

```
public Operand copy()
```

Description copied from class: DrawableComponent

Deep copy method for DrawableComponents. The new component has the same coordinates and the same number of DockingPoints. Hooked GateConnections won't be copied.

Overrides:

copy in class DrawableComponent

Returns:

A deep copy of the DrawableComponent.

---

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

Description copied from class: DrawableComponent

Sets the bounds of the DrawableComponent.

Overrides:

setBounds in class DrawableComponent

Parameters:

x - The new x coordinate of the DrawableComponent.

y - The new y coordinate of the DrawableComponent.

width - The new width of the DrawableComponent.

height - The new height of the DrawableComponent.

Package

domain.qualifyers

Class Summary		Page
BooleanQualifyer	Represents an element of a Boolean function.	93
ButtonQualifyer	An element that can be passed to a `QualifiedToggleButton` containing information that is displayed in the button.	94
QualifyerGate	Represents an operator or a NOT sign in a Boolean function.	95
QualifyerOperand	Represents an operand in a Boolean function.	97

domain.qualifyers
BooleanQualifyer

```
java.lang.Object
  └ domain.qualifyers.ButtonQualifyer
      └ domain.qualifyers.BooleanQualifyer
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

QualifyerGate, QualifyerOperand

```
public abstract class BooleanQualifyer
extends ButtonQualifyer
```

Represents an element of a Boolean function. Used to define such an element, that a `DrawableComponent` or a button stands for.

Constructor Summary	Page
BooleanQualifyer`(char c, ImageIcon buttonIcon, ImageIcon selectedButtonIcon)` Constructs a new `BooleanQualifyer`.	93

Method Summary		Page
boolean	**equals**`(Object o)` Returns `True` if the passed object represents the same part of a Boolean function.	94
char	**getChar**`()` Returns the character of an element in a Boolean function that is represented by this `BooleanQualifyer`.	94

Constructor Detail

BooleanQualifyer

```
public BooleanQualifyer(char c,
                        ImageIcon buttonIcon,
                        ImageIcon selectedButtonIcon)
```

Constructs a new `BooleanQualifyer`.

Parameters:

`c` - A character of an element in a Boolean function that is represented.

`buttonIcon` - The icon of a button that is associated with this Boolean element.

`selectedButtonIcon` - The selected icon of a button that is associated with this Boolean element.

Method Detail

getChar

`public char` **`getChar`**`()`

Returns the character of an element in a Boolean function that is represented by this `BooleanQualifyer`.

Returns:

The character of an element in a Boolean function that is represented by this `BooleanQualifyer`.

---

equals

`public boolean` **`equals`**`(Object o)`

Returns `True` if the passed object represents the same part of a Boolean function.

Overrides:

`equals` in class `Object`

Parameters:

`o` - The object that is to be compared to this object.

Returns:

`True` if the passed object represents the same part of a Boolean function. `False` otherwise.

domain.qualifyers
ButtonQualifyer

```
java.lang.Object
  └ domain.qualifyers.ButtonQualifyer
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

BooleanQualifyer

```
public class ButtonQualifyer
extends Object
implements Serializable
```

An element that can be passed to a `QualifiedToggleButton` containing information that is displayed in the button.

Constructor Summary	Page
ButtonQualifyer(ImageIcon buttonIcon, ImageIcon selectedButtonIcon) Constructs a new `ButtonQualifyer`.	95

Method Summary		Page
ImageIcon	**getButtonIcon**() Returns the icon of the button.	95
ImageIcon	**getSelectedButtonIcon**() Returns the selected icon of the button.	95

Constructor Detail

ButtonQualifyer

```
public ButtonQualifyer(ImageIcon buttonIcon,
                       ImageIcon selectedButtonIcon)
```

Constructs a new `ButtonQualifyer`.

Parameters:

`buttonIcon` - The icon of the button.

`selectedButtonIcon` - The selected icon of the button.

Method Detail

getButtonIcon

```
public ImageIcon getButtonIcon()
```

Returns the icon of the button.

Returns:

The icon of the button.

---

getSelectedButtonIcon

```
public ImageIcon getSelectedButtonIcon()
```

Returns the selected icon of the button.

Returns:

The selected icon of the button.

domain.qualifyers
QualifyerGate

```
java.lang.Object
  └ domain.qualifyers.ButtonQualifyer
      └ domain.qualifyers.BooleanQualifyer
          └ domain.qualifyers.QualifyerGate
```

All Implemented Interfaces:

Serializable

```
public class QualifyerGate
extends BooleanQualifyer
```

Represents an operator or a NOT sign in a Boolean function.

Constructor Summary	Page
QualifyerGate(char c, ImageIcon GateIcon, ImageIcon buttonIcon, ImageIcon, selectedIcon) Constructs a new QualifyerGate.	96

Method Summary		Page
Image	**getGateIcon**() Returns the icon of the gate that this qualifyer represents.	96
void	**setSymbol**(ImageIcon symbol) Sets the icon of the gate that this qualifyer represents.	96

Constructor Detail

QualifyerGate

```
public QualifyerGate(char c,
                     ImageIcon GateIcon,
                     ImageIcon buttonIcon,
                     ImageIcon selectedButtonIcon)
```

Constructs a new QualifyerGate.

Parameters:

c - A character of an Boolean operator that this qualifyer represents.

GateIcon - Icon of the gate that this qualifyer represents.

buttonIcon - The icon of a button that is associated with this Boolean operator.

selectedButtonIcon - The selected icon of a button that is associated with this Boolean operator.

Method Detail

getGateIcon

```
public Image getGateIcon()
```

Returns the icon of the gate that this qualifyer represents.

Returns:

The icon of the gate that this qualifyer represents.

---

setSymbol

```
public void setSymbol(ImageIcon symbol)
```

Sets the icon of the gate that this qualifyer represents.

Parameters:

symbol - The icon of the gate that this qualifyer represents.

domain.qualifyers
QualifyerOperand

```
java.lang.Object
  └ domain.qualifyers.ButtonQualifyer
      └ domain.qualifyers.BooleanQualifyer
          └ domain.qualifyers.QualifyerOperand
```

All Implemented Interfaces:

Serializable

```
public class QualifyerOperand
extends BooleanQualifyer
```

Represents an operand in a Boolean function. Defines the truth value of this Boolean operand for the whole program.

Constructor Summary	Page
QualifyerOperand(char operand, ImageIcon icon, ImageIcon selectedButtonIcon) Constructs a new QualifyerOperand.	97

Method Summary		Page
boolean	**getValue**() Returns the truth value of the operand that is represented by this qualifyer.	97
void	**setValue**(boolean v) Sets the truth value of the operand that is represented by this qualifyer.	97

Constructor Detail

QualifyerOperand

```
public QualifyerOperand(char operand,
                        ImageIcon icon,
                        ImageIcon selectedButtonIcon)
```

Constructs a new QualifyerOperand.

Parameters:

operand - A character of an Boolean operand that this qualifyer represents.

icon - The icon of a button that is associated with this Boolean operator.

selectedButtonIcon - The selected icon of a button that is associated with this Boolean operator.

Method Detail

setValue

```
public void setValue(boolean v)
```

Sets the truth value of the operand that is represented by this qualifyer.

Parameters:

v - The truth value of the operand that is represented by this qualifyer.

---

getValue

```
public boolean getValue()
```

Returns the truth value of the operand that is represented by this qualifyer.

Returns:

The truth value of the operand that is represented by this qualifyer.

Package

gui

Class Summary		Page
ButtonArea	The toolbox of the program providing symbols for the elements that can be drawn on the DrawingArea as well as buttons for different functions and actions that can be performed.	98
Dialog	Shows a dialog window with a message.	100
DrawingArea	The area of the program, where logic circuits are drawn.	101
FileDialog	Shows a dialog window for choosing a file.	104
GateDrawer	The main window of the GUI.	106
GlobalKeyListener	Keylistener for the whole program.	114
Main	The Main class of this project that instantiates a controller to start the program.	116
QualifiedToggleButton	A JToggleButton taking its data from a Qualifyer.	117

gui

ButtonArea

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └gui.ButtonArea
```

All Implemented Interfaces:

ActionListener, EventListener, ItemListener, Accessible, Serializable, ImageObserver, MenuContainer

```
public class ButtonArea
extends JPanel
implements ActionListener, ItemListener
```

The toolbox of the program providing symbols for the elements that can be drawn on the DrawingArea as well as buttons for different functions and actions that can be performed.

Constructor Summary	Page
ButtonArea(GateDrawer parent, Controller controller) Constructs a new ButtonArea.	99

Method Summary		Page
void	actionPerformed(ActionEvent event) Handles action events when a button in the toolbox is clicked.	99
void	itemStateChanged(ItemEvent arg0) Updates the truth values of all operands after one of them has changed in the toolbox.	99
void	refreshValues() Sets the truth value of all operands to those specified by the user in the toolbox.	99
void	setButtonsEnabled(boolean b) Enables or disables all buttons that have nothing to do with the drawing of logic circuit in the toolbox.	99
void	setDrawingInputEnabled(boolean b) Enables or disables all buttons that have to do with the drawing of logic circuit in the toolbox.	99

Constructor Detail

ButtonArea

```
public ButtonArea(GateDrawer parent,
                  Controller controller)
```

Constructs a new `ButtonArea`.

Parameters:

`parent` - The parent GUI containing this toolbox.

`controller` - The controller of this program.

Method Detail

refreshValues

```
public void refreshValues()
```

Sets the truth value of all operands to those specified by the user in the toolbox.

---

actionPerformed

```
public void actionPerformed(ActionEvent event)
```

Handles action events when a button in the toolbox is clicked. Operations are passed to the parent GUI or directly to the controller.

Specified by:

`actionPerformed` in interface `ActionListener`

---

setButtonsEnabled

```
public void setButtonsEnabled(boolean b)
```

Enables or disables all buttons that have nothing to do with the drawing of logic circuit in the toolbox.

Parameters:

`b` - `True` if all buttons not regarding drawing of logic circuits should be enabled. `False` otherwise.

---

setDrawingInputEnabled

```
public void setDrawingInputEnabled(boolean b)
```

Enables or disables all buttons that have to do with the drawing of logic circuit in the toolbox.

Parameters:

`b` - `True` if all buttons regarding drawing of logic circuits should be enabled. `False` otherwise.

---

itemStateChanged

```
public void itemStateChanged(ItemEvent arg0)
```

Updates the truth values of all operands after one of them has changed in the toolbox.

Specified by:

`itemStateChanged` in interface `ItemListener`

gui
Dialog

```
java.lang.Object
  └ gui.Dialog
```

```
public class Dialog
extends Object
```

Shows a dialog window with a message.

Constructor Summary	Page
Dialog()	100

Method Summary		Page
static boolean	ask(String message, String title) Shows a question dialog with yes/no-option for the user.	100
static void	info() Shows a window with information about the program.	101
static void	tellError(String message, String title) Shows an error message with a given text.	100
static void	tellSuccess(String message, String title) Show an information window to display a success.	100

Constructor Detail

Dialog

```
public Dialog()
```

Method Detail

tellError

```
public static void tellError(String message,
                             String title)
```

Shows an error message with a given text.

Parameters:

`message` - Message to display.

`title` - Title of the error window.

---

tellSuccess

```
public static void tellSuccess(String message,
                               String title)
```

Show an information window to display a success.

Parameters:

`message` - Message to display.

`title` - Title of the information window.

---

ask

```
public static boolean ask(String message,
                          String title)
```

Shows a question dialog with yes/no-option for the user.

Parameters:

`message` - Message to display.

`title` - Title of the dialog.

Returns:

`True` if the user chose yes. `False` otherwise.

---

info

```
public static void info()
```

Shows a window with information about the program.

gui
DrawingArea

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └gui.DrawingArea
```

All Implemented Interfaces:

Accessible, Serializable, ImageObserver, MenuContainer

```
public class DrawingArea
extends JPanel
```

The area of the program, where logic circuits are drawn. All calculations and changes are performed in `DrawingService`. `DrawingArea` is just displaying the logic circuit.

Constructor Summary	Page
DrawingArea(Controller controller) Constructs an new DrawingArea	102

Method Summary		Page
protected void	**paintComponent**(Graphics g)	103
void	**removeCurrentNode**() Removes the Node that is currently displayed.	103
void	**removePopup**() Removes the current popup.	104
void	**removeSelection**() Removes the current selection.	103
void	**setCurrentLine**(GateConnection con) Sets a GateConnection that is currently constructed by the user and should be displayed.	103
void	**setCurrentNode**(Node node) Sets the current node that should be displayed.	103
void	**setGraph**(Graph graph) Sets the Graph that should be displayed.	103
void	**setGrid**(GridService grid) Sets the grid of the DrawingArea.	103
void	**setSelection**(Rectangle selection) Sets a selected area in the DrawingArea.	103
void	**showComponentPopup**(DrawableComponent dc) Shows a popup with options to modify a component.	104
void	**showConnectionPopup**(Node currentNode) Shows a popup with options to modify the connection of the current node.	104
void	**showSelectionPopup**() Shows a popup to modify the current selection.	104

Constructor Detail

DrawingArea

```
public DrawingArea(Controller controller)
```

Constructs an new DrawingArea

Parameters:

`controller` - The controller of this program.

Method Detail

setGraph

`public void` **`setGraph`**`(Graph graph)`

Sets the `Graph` that should be displayed.

Parameters:

`graph` - The `Graph` that should be displayed.

---

setGrid

`public void` **`setGrid`**`(GridService grid)`

Sets the grid of the `DrawingArea`.

Parameters:

`grid` - The new grid of the `DrawingArea`.

---

setCurrentNode

`public void` **`setCurrentNode`**`(Node node)`

Sets the current node that should be displayed.

Parameters:

`node` - The `Node` to display.

---

removeCurrentNode

`public void` **`removeCurrentNode`**`()`

Removes the `Node` that is currently displayed.

---

setCurrentLine

`public void` **`setCurrentLine`**`(GateConnection con)`

Sets a `GateConnection` that is currently constructed by the user and should be displayed.

Parameters:

`con` - The `GateConnection` to display.

---

setSelection

`public void` **`setSelection`**`(Rectangle selection)`

Sets a selected area in the `DrawingArea`. The selection is drawn as a rectangle.

Parameters:

`selection` - `Rectangle` defining the selected area.

---

removeSelection

`public void` **`removeSelection`**`()`

Removes the current selection.

---

paintComponent

`protected void` **`paintComponent`**`(Graphics g)`

Overrides:

`paintComponent` in class `JComponent`

---

showConnectionPopup

`public void` **`showConnectionPopup`**`(Node currentNode)`

Shows a popup with options to modify the connection of the current node.

Parameters:

`currentNode` - The `Node` to edit.

---

showComponentPopup

`public void` **`showComponentPopup`**`(DrawableComponent dc)`

Shows a popup with options to modify a component.

Parameters:

`dc` - The `DrawableComponent` to modify.

---

showSelectionPopup

`public void` **`showSelectionPopup`**`()`

Shows a popup to modify the current selection.

---

removePopup

`public void` **`removePopup`**`()`

Removes the current popup.

gui
FileDialog

```
java.lang.Object
  └ gui.FileDialog
```

```
public class FileDialog
extends Object
```

Shows a dialog window for choosing a file.

Constructor Summary	Page
`FileDialog()`	105

Method Summary		Page
static File	**showExportDialog**() Shows a file chooser for exporting an image with file extension "jpeg" or "jpg".	105
static File	**showLoadDialog**(FileNameExtensionFilter filter) Shows a file chooser for loading a file with a given file extension.	105
static File	**showSaveDialog**(FileNameExtensionFilter filter) Shows a file chooser for saving a file with a given file extension.	105

Constructor Detail

FileDialog

```
public FileDialog()
```

Method Detail

showSaveDialog

```
public static File showSaveDialog(FileNameExtensionFilter filter)
```

Shows a file chooser for saving a file with a given file extension. If the chosen file already exists the user is asked whether he wants to overwrite it.

Parameters:

`filter` - The file extension the file must have.

Returns:

`The chosen file` or `null` if the user chose nothing.

---

showLoadDialog

```
public static File showLoadDialog(FileNameExtensionFilter filter)
```

Shows a file chooser for loading a file with a given file extension.

Parameters:

`filter` - The file extension the file must have.

Returns:

`The chosen file` or `null` if the user chose nothing.

---

showExportDialog

```
public static File showExportDialog()
```

Shows a file chooser for exporting an image with file extension "jpeg" or "jpg". If the chosen file already exists the user is asked whether he wants to overwrite it.

Returns:

`The chosen file` or `null` if the user chose nothing.

gui
GateDrawer

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └gui.GateDrawer
```

All Implemented Interfaces:

WindowListener, EventListener, ComponentListener, WindowConstants, Accessible, RootPaneContainer, MenuContainer, ImageObserver, Serializable

---

```
public class GateDrawer
extends JFrame
implements WindowListener, ComponentListener
```

The main window of the GUI. Instantiates all the parts of the GUI and works as an interface between GUI and Controller.

Constructor Summary	Page
GateDrawer(Controller controller) Constructs a new GateDrawer.	108

	Method Summary	Page
void	**abortExercise**() Aborts the current Exercise - either creation or execution or displaying of result.	110
void	**clearGraph**() Removes the current logic circuit.	113
void	**clearGraphFunction**() Clears the graph function textfield.	112
void	**clearTextFields**() Clears all textfields.	112
void	**closeExerciseMenuBar**() Closes the current ExerciseMenuBar.	113
void	**compareFunctionAndGraph**() Checks if the current logic circuit and the current user-entered Boolean function are equal.	113
void	**componentHidden**(ComponentEvent arg0) ComponentHidden event is ignored.	114
void	**componentMoved**(ComponentEvent arg0) ComponentMoved event is ignored.	114
void	**componentResized**(ComponentEvent arg0) On a ComponentResized event the font sizes of all textfields are adjusted.	114
void	**componentShown**(ComponentEvent arg0) ComponentShown event is ignored.	114
void	**createExercises**() Starts the creation of a new set of Exercises.	110
void	**deleteExercise**() Deletes the current Exercise.	110
void	**display**(String text) Displays a message in jtaText.	113
void	**doExercises**() Loads a new set of Exercises for execution.	111

void	**editExercises**() Loads a file of Exercises to edit them.	113
void	**evaluateExercise**() Evaluates the current Exercise.	111
void	**executeCurrentExercise**() Starts execution of the current Exericse.	111
void	**exit**() Saves the current state of the program and exits.	109
void	**export**() Exports a the current logic circuit as an image.	109
void	**finishExerciseCreation**() Lets user choose a file where the current Exercise should be stored in and tells the ExerciseService to save the Exercise.	111
void	**finishExerciseExecution**() Saves a set of Exercises after a user executed them.	113
String	**generateFunction**() Shows a randomly generated Boolean function.	112
DrawingArea	**getDrawingArea**() Returns the DrawingArea of the GUI.	111
String	**getEnteredFunction**() Returns the Boolean function that the user has entered.	111
void	**giveUpShowSolution**() The user gives up the current Exercise and the correct solution is shown.	113
void	**isFunctionSatisfiable**() Tests if the current user-entered Boolean function is satisfiable and displays the result.	114
void	**isGraphSatisfiable**() Tests if the current logic circuit is satisfiable and displays the result.	114
void	**load**() Load a program state from a file.	109
void	**loadResults**() Loads a file with already executed Exercises and shows the results.	113
void	**newExercise**() Tells ExerciseService to create a new (empty) Exercise.	110
void	**refreshValues**() Refreshes the truth values of all operands to those that the user chose in the ButtonArea.	111
void	**removeExerciseMenuBar**() Removes the current ExerciseMenuBar from the GUI, packs and repaints the GUI.	110
void	**save**() Shows a file dialogue to choose a file where the current state of the program should be saved to.	112
void	**saveExercise**(ExerciseType mode, int points, int attempts) Saves the Exercise that is currently displayed while creating a new set of Exercises Creates a new ExerciseService if necessary.	110
void	**setGateFunction**(String function) Sets the function equal to the logic circuit.	112
void	**setUserFunction**(String function) Sets the user entered Boolean function to a given function.	112
void	**showCorrectSolution**() Displays the correct solution for the current Exercise.	111
void	**showCurrentExercise**(boolean showSolution) Displays the current Exercise.	110

void	**showGeneratedGraph**() Shows a randomly generated logic circuit.	113
void	**showGraphAsFunction**() Shows Boolean function equal to the current logic circuit.	112
void	**showInfo**() Shows a window with information about the program.	114
void	**updateEvaluations**() Updates the evaluations of the user-entered Boolean function and the logic circuit.	114
void	**userFunctionToGraph**() Shows a logic circuit equal to the current user-entered Boolean function.	112
void	**windowActivated**(WindowEvent arg0) WindowActivated Event is ignored.	108
void	**windowClosed**(WindowEvent arg0) WindowClosed Event is ignored.	108
void	**windowClosing**(WindowEvent arg0) Calls exit() in GateDrawer.	109
void	**windowDeactivated**(WindowEvent arg0) WindowDeactivated Event is ignored.	109
void	**windowDeiconified**(WindowEvent arg0) WindowDeiconified Event is ignored.	109
void	**windowIconified**(WindowEvent arg0) WindowIconified Event is ignored.	109
void	**windowOpened**(WindowEvent arg0) WindowOpened Event is ignored.	109

Constructor Detail

GateDrawer

public **GateDrawer**(Controller controller)

Constructs a new GateDrawer. Initializes all GUI elements and aligns and displays them.

Parameters:

controller - The main Controller of the program.

Method Detail

windowActivated

public void **windowActivated**(WindowEvent arg0)

WindowActivated Event is ignored.

Specified by:

windowActivated in interface WindowListener

---

windowClosed

public void **windowClosed**(WindowEvent arg0)

WindowClosed Event is ignored.

Specified by:

windowClosed in interface WindowListener

---

windowClosing

`public void` **`windowClosing`**`(WindowEvent arg0)`

Calls `exit()` in `GateDrawer`.

Specified by:

`windowClosing` in interface `WindowListener`

---

windowDeactivated

`public void` **`windowDeactivated`**`(WindowEvent arg0)`

WindowDeactivated Event is ignored.

Specified by:

`windowDeactivated` in interface `WindowListener`

---

windowDeiconified

`public void` **`windowDeiconified`**`(WindowEvent arg0)`

WindowDeiconified Event is ignored.

Specified by:

`windowDeiconified` in interface `WindowListener`

---

windowIconified

`public void` **`windowIconified`**`(WindowEvent arg0)`

WindowIconified Event is ignored.

Specified by:

`windowIconified` in interface `WindowListener`

---

windowOpened

`public void` **`windowOpened`**`(WindowEvent arg0)`

WindowOpened Event is ignored.

Specified by:

`windowOpened` in interface `WindowListener`

---

exit

`public void` **`exit`**`()`

Saves the current state of the program and exits. The state is saved in "autosave.gser" in the directory that the program is running in. Saving is done in `Controller`.

---

export

`public void` **`export`**`()`

Exports a the current logic circuit as an image. Opens a file dialogue for the user to choose a file. Export is done in `Controller`.

---

load

`public void` **`load`**`()`

Load a program state from a file. Opens a file dialogue for the user to choose a file. Loading is done in `Controller`.

---

createExercises

`public void` **`createExercises`**`()`

Starts the creation of a new set of `Exercise`s. Therefore all current `Exercise`-related activities are aborted. A new `ExerciseService` is instantiated and a `CreateExerciseMenuBar` is shown.

---

removeExerciseMenuBar

`public void` **`removeExerciseMenuBar`**`()`

Removes the current `ExerciseMenuBar` from the GUI, packs and repaints the GUI.

---

abortExercise

`public void` **`abortExercise`**`()`

Aborts the current `Exercise` - either creation or execution or displaying of result. User is asked if the `Exercise` should be aborted. Only if the user wishes to, the `Exercise` is aborted.

---

saveExercise

```
public void saveExercise(ExerciseType mode,
                         int points,
                         int attempts)
```

Saves the `Exercise` that is currently displayed while creating a new set of `Exercise`s Creates a new `ExerciseService` if necessary.

Parameters:

`mode` - The type of the `Exercise`.

`points` - Points that this `exercise` is worth.

`attempts` - The number of attempts that may be made while solving this task.

---

showCurrentExercise

`public void` **`showCurrentExercise`**`(boolean showSolution)`

Displays the current `Exercise`. According to its type, a function, gates or both are shown.

Parameters:

`showSolution` - `True` if the users solution should be displayed. `False` otherwise.

---

newExercise

`public void` **`newExercise`**`()`

Tells `ExerciseService` to create a new (empty) `Exercise`. A new `Exercise` is created, if the last `Exercise` is not empty. All fields and the `Graph are cleared so that an empty GUI is shown.`

---

deleteExercise

`public void` **`deleteExercise`**`()`

Deletes the current `Exercise`. The previous `Exercise is shown (or the first if no previous Exercise exists).`

---

finishExerciseCreation

`public void` **`finishExerciseCreation`**`()`

Lets user choose a file where the current `Exercise` should be stored in and tells the `ExerciseService` to save the `Exercise`. `ExerciseCreationMenuBar` is closed afterwards.

---

evaluateExercise

`public void` **`evaluateExercise`**`()`

Evaluates the current `Exercise`. Evaluation is done in `ExerciseService`.

---

doExercises

`public void` **`doExercises`**`()`

Loads a new set of `Exercise`s for execution. `Exercise`s are loaded from a file that the user chooses and a new `ExerciseService` and `ExerciseMenu` are instantiated.

---

executeCurrentExercise

`public void` **`executeCurrentExercise`**`()`

Starts execution of the current `Exericse`. Enables or disables buttons, fields and drawing according to the type of the `Exercise`.

---

showCorrectSolution

`public void` **`showCorrectSolution`**`()`

Displays the correct solution for the current `Exercise`.

---

getDrawingArea

`public` `DrawingArea` **`getDrawingArea`**`()`

Returns the `DrawingArea` of the GUI.

Returns:

The `DrawingArea` of the GUI.

---

getEnteredFunction

`public String` **`getEnteredFunction`**`()`

Returns the Boolean function that the user has entered.

Returns:

The Boolean function that the user has entered.

---

refreshValues

`public void` **`refreshValues`**`()`

Refreshes the truth values of all operands to those that the user chose in the `ButtonArea`.

---

save

`public void` **`save`**`()`

Shows a file dialogue to choose a file where the current state of the program should be saved to. Saving is done in `Controller`.

---

setUserFunction

`public void` **`setUserFunction`**`(String function)`

Sets the user entered Boolean function to a given function.

Parameters:

`function` - a Boolean function.

---

setGateFunction

`public void` **`setGateFunction`**`(String function)`

Sets the function equal to the logic circuit.

Parameters:

`function` - A Boolean function equal to the logic circuit.

---

clearTextFields

`public void` **`clearTextFields`**`()`

Clears all textfields.

---

clearGraphFunction

`public void` **`clearGraphFunction`**`()`

Clears the graph function textfield.

---

showGraphAsFunction

`public void` **`showGraphAsFunction`**`()`

Shows Boolean function equal to the current logic circuit.

---

userFunctionToGraph

`public void` **`userFunctionToGraph`**`()`

Shows a logic circuit equal to the current user-entered Boolean function.

---

generateFunction

`public String` **`generateFunction`**`()`

Shows a randomly generated Boolean function. Generation is done in `Controller`.

Returns:

A randomly generated Boolean function.

---

compareFunctionAndGraph

`public void` **`compareFunctionAndGraph`**`()`

Checks if the current logic circuit and the current user-entered Boolean function are equal. The result is displayed.

---

display

`public void` **`display`**`(String text)`

Displays a message in `jtaText`.

Parameters:

`text` - Message to display.

---

showGeneratedGraph

`public void` **`showGeneratedGraph`**`()`

Shows a randomly generated logic circuit. Generation is done in `Controller`.

---

clearGraph

`public void` **`clearGraph`**`()`

Removes the current logic circuit.

---

giveUpShowSolution

`public void` **`giveUpShowSolution`**`()`

The user gives up the current `Exercise` and the correct solution is shown.

---

editExercises

`public void` **`editExercises`**`()`

Loads a file of `Exercise`s to edit them.

---

loadResults

`public void` **`loadResults`**`()`

Loads a file with already executed `Exercise`s and shows the results.

---

finishExerciseExecution

`public void` **`finishExerciseExecution`**`()`

Saves a set of `Exercise`s after a user executed them. The `Exercise`s are saved to the same file as they were loaded from. If the original file doesn't exist anymore, the user can choose another one.

---

closeExerciseMenuBar

`public void` **`closeExerciseMenuBar`**`()`

Closes the current `ExerciseMenuBar`. User is asked whether to save the current state.

---

showInfo

`public void` **`showInfo`**`()`

Shows a window with information about the program.

---

updateEvaluations

`public void` **`updateEvaluations`**`()`

Updates the evaluations of the user-entered Boolean function and the logic circuit.

---

isFunctionSatisfiable

`public void` **`isFunctionSatisfiable`**`()`

Tests if the current user-entered Boolean function is satisfiable and displays the result.

---

isGraphSatisfiable

`public void` **`isGraphSatisfiable`**`()`

Tests if the current logic circuit is satisfiable and displays the result.

---

componentHidden

`public void` **`componentHidden`**`(ComponentEvent arg0)`

ComponentHidden event is ignored.

Specified by:

`componentHidden` in interface `ComponentListener`

---

componentMoved

`public void` **`componentMoved`**`(ComponentEvent arg0)`

ComponentMoved event is ignored.

Specified by:

`componentMoved` in interface `ComponentListener`

---

componentResized

`public void` **`componentResized`**`(ComponentEvent arg0)`

On a ComponentResized event the font sizes of all textfields are adjusted.

Specified by:

`componentResized` in interface `ComponentListener`

---

componentShown

`public void` **`componentShown`**`(ComponentEvent arg0)`

ComponentShown event is ignored.

Specified by:

`componentShown` in interface `ComponentListener`

gui
GlobalKeyListener

```
java.lang.Object
  └ gui.GlobalKeyListener
```

All Implemented Interfaces:

KeyListener, EventListener, Serializable

```
public class GlobalKeyListener
extends Object
implements KeyListener, Serializable
```

Keylistener for the whole program. Responsible for changing specific keys to operator signs.

Constructor Summary	Page
GlobalKeyListener(Controller controller, GateDrawer gui) Constructs a new GlobalKeyListener.	115

Method Summary		Page
void	**keyPressed**(KeyEvent e) KeyPressed event is ignored.	115
void	**keyReleased**(KeyEvent ke) When a key is released it is checked for ESC or DEL key.	115
void	**keyTyped**(KeyEvent ke) When a key is typed, its code is checked and may be changed to another one, depending on its character.	116

Constructor Detail

GlobalKeyListener

```
public GlobalKeyListener(Controller controller,
                         GateDrawer gui)
```

Constructs a new GlobalKeyListener.

Parameters:

controller - The Controller of the program.

gui - The GUI of the program.

Method Detail

keyPressed

```
public void keyPressed(KeyEvent e)
```

KeyPressed event is ignored.

Specified by:

keyPressed in interface KeyListener

---

keyReleased

`public void` **`keyReleased`**`(KeyEvent ke)`

When a key is released it is checked for `ESC` or `DEL` key.

- ESC: the drawing of the current line is aborted.
- DEL: the current selection in DrawingArea is deleted.

Specified by:

`keyReleased` in interface `KeyListener`

---

keyTyped

`public void` **`keyTyped`**`(KeyEvent ke)`

When a key is typed, its code is checked and may be changed to another one, depending on its character.

- `+` becomes AND.
- `-` becomes OR.
- `/` becomes NOT.
- `x` becomes XOR.
- `*` (`SHIFT` + `+`) becomes NAND.
- `_` (`SHIFT` + `-`) becomes NOR.

Specified by:

`keyTyped` in interface `KeyListener`

gui
Main

```
java.lang.Object
  └ gui.Main
```

```
public class Main
extends Object
```

The Main class of this project that instantiates a controller to start the program.

Constructor Summary	Page
Main()	117

Method Summary		Page
static void	main(String[] args) Instantiates a new Controller to start the program.	117

Constructor Detail

Main

```
public Main()
```

Method Detail

main

```
public static void main(String[] args)
```

Instantiates a new Controller to start the program.

Parameters:

`args` - Parameters are ignored.

gui
QualifiedToggleButton

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.AbstractButton
                  └javax.swing.JToggleButton
                      └gui.QualifiedToggleButton
```

All Implemented Interfaces:

Accessible, ItemSelectable, SwingConstants, Serializable, ImageObserver, MenuContainer

---

`public class` **`QualifiedToggleButton`**
`extends JToggleButton`

A JToggleButton taking its data from a `Qualifyer`.

Constructor Summary	Page
`QualifiedToggleButton``(ButtonQualifyer qualifyer)` Constructs a new `QualifiedToggleButton`.	118

Method Summary		Page
`ButtonQualifyer`	**`getQualifyer`**`()` Returns the `Qualifyer` that defines the button.	118

Constructor Detail

QualifiedToggleButton

`public` **`QualifiedToggleButton`**`(ButtonQualifyer qualifyer)`

Constructs a new `QualifiedToggleButton`.

Parameters:

`qualifyer` - The `Qualifyer` defining the button.

Method Detail

getQualifyer

`public ButtonQualifyer` **`getQualifyer`**`()`

Returns the `Qualifyer` that defines the button.

Returns:

The `Qualifyer` that defines the button.

Package

gui.container

Class Summary		Page
FunctionContainer	A Container for 4 different GUI elements: a textfield for in- or output of Boolean functions, three buttons for generating a random function or circuit, convert a circuit in a function or vice versa and deleting the function or a circuit.	119
GraphFunctionContainer	Container for Boolean function and some buttons for the logic circuit displayed in `DrawingArea`.	122
RoundEdgeContainer	Contains a part of the GUI and surrounds it with a round-edged, colored rectangle.	124
UserFunctionContainer		125

gui.container
FunctionContainer

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └gui.container.FunctionContainer
```

All Implemented Interfaces:

ActionListener, EventListener, Accessible, Serializable, ImageObserver, MenuContainer

Direct Known Subclasses:

GraphFunctionContainer, UserFunctionContainer

```
public abstract class FunctionContainer
extends JPanel
implements ActionListener
```

A Container for 4 different GUI elements: a textfield for in- or output of Boolean functions, three buttons for generating a random function or circuit, convert a circuit in a function or vice versa and deleting the function or a circuit. The exact functionality is defined in subclasses and a label for displaying the truth value of the function or circuit.

Field Summary		Page
protected JPanel	**buttonPanel** Contains the buttons.	120
protected Font	**evalFont** Font for `lbEval`.	120
protected Font	**functionFont** Font for `jtfFunction`.	120
protected JPanel	**functionPanel** Contains the textfield and the label.	120
protected JButton	**jbClear** Clears a function or circuit.	120
protected JButton	**jbDisplay** Displays a function as a circuit or vice versa.	121

protected JButton	**jbRandom** Generates a random Boolean function or logic circuit.	121
protected JTextField	**jtfFunction** Textfield displaying a Boolean function.	121
protected JLabel	**lbEval** Displays the truth value of a function or a circuit.	121
protected GateDrawer	**parent** The GUI of the program.	121

Constructor Summary	Page
FunctionContainer(GateDrawer parent, KeyListener keyListener) Constructs a new FunctionContainer.	121

Method Summary		Page
String	**getFunction**() Returns the Boolean function that is displayed.	122
protected abstract void	**initializeComponent**() Initializations depending of the subtype.	121
void	**setButtonsEnabled**(boolean b) Enables the jbDisplay and the jbRandom buttons.	122
void	**setEnabled**(boolean b) Sets all elements of this container enabled or disabled.	121
void	**setEval**(boolean b) Displays a truth value in lbEval.	122
void	**setFontSizes**(int size) Sets the font size of the textfield where the Boolean function is displayed.	122
void	**setFunction**(String function) Displays a Boolean function in jtfFunction.	122

Field Detail

buttonPanel

protected JPanel **buttonPanel**

Contains the buttons.

evalFont

protected Font **evalFont**

Font for lbEval.

functionFont

protected Font **functionFont**

Font for jtfFunction.

functionPanel

protected JPanel **functionPanel**

Contains the textfield and the label.

jbClear

protected JButton **jbClear**

Clears a function or circuit.

jbDisplay

```
protected JButton jbDisplay
```

Displays a function as a circuit or vice versa.

---

jbRandom

```
protected JButton jbRandom
```

Generates a random Boolean function or logic circuit.

---

jtfFunction

```
protected JTextField jtfFunction
```

Textfield displaying a Boolean function.

---

lbEval

```
protected JLabel lbEval
```

Displays the truth value of a function or a circuit.

---

parent

```
protected GateDrawer parent
```

The GUI of the program.

Constructor Detail

FunctionContainer

```
public FunctionContainer(GateDrawer parent,
                         KeyListener keyListener)
```

Constructs a new `FunctionContainer`.

Parameters:

`parent` - The GUI of the program.

`keyListener` - The keylistener for changing normal keys into operator signs.

Method Detail

initializeComponent

```
protected abstract void initializeComponent()
```

Initializations depending of the subtype.

---

setEnabled

```
public void setEnabled(boolean b)
```

Sets all elements of this container enabled or disabled.

Overrides:

`setEnabled` in class `JComponent`

Parameters:

`b` - `True` if all elements of this container should be enabled. `False` otherwise.

---

setFunction

`public void` **`setFunction`**`(String function)`

Displays a Boolean function in `jtfFunction`.

Parameters:

`function` - The Boolean function that should be displayed.

---

setEval

`public void` **`setEval`**`(boolean b)`

Displays a truth value in `lbEval`.

Parameters:

`b` - `True` if "wahr" should be displayed. `False` if "falsch" should be displayed.

---

getFunction

`public String` **`getFunction`**`()`

Returns the Boolean function that is displayed.

Returns:

The Boolean function that is displayed.

---

setButtonsEnabled

`public void` **`setButtonsEnabled`**`(boolean b)`

Enables the `jbDisplay` and the `jbRandom` buttons.

Parameters:

`b` - `True` if the buttons should be enabled. `False` otherwise.

---

setFontSizes

`public void` **`setFontSizes`**`(int size)`

Sets the font size of the textfield where the Boolean function is displayed.

Parameters:

`size` - The new font size of the textfield.

gui.container
GraphFunctionContainer

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └gui.container.FunctionContainer
                      └gui.container.GraphFunctionContainer
```

All Implemented Interfaces:

ActionListener, EventListener, Accessible, Serializable, ImageObserver, MenuContainer

---

```
public class GraphFunctionContainer
extends FunctionContainer
```

Container for Boolean function and some buttons for the logic circuit displayed in `DrawingArea`.

Constructor Summary	Page
GraphFunctionContainer(GateDrawer parent, KeyListener keyListener) Constructs a new `GraphFunctionContainer`.	123

Method Summary		Page
void	**actionPerformed**(ActionEvent e) Handles action events on the buttons in this container.	123
protected void	**initializeComponent**() Initializations depending of the subtype.	123

Constructor Detail

GraphFunctionContainer

```
public GraphFunctionContainer(GateDrawer parent,
                              KeyListener keyListener)
```

Constructs a new `GraphFunctionContainer`.

Parameters:

`parent` - The parent GUI holding this container.

`keyListener` - A keylistener for formula input.

Method Detail

initializeComponent

```
protected void initializeComponent()
```

Description copied from class: FunctionContainer

Initializations depending of the subtype.

Overrides:

`initializeComponent` in class `FunctionContainer`

---

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

Handles action events on the buttons in this container. Actions are passed to the parent GUI.

Specified by:

`actionPerformed` in interface `ActionListener`

gui.container
RoundEdgeContainer

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └gui.container.RoundEdgeContainer
```

All Implemented Interfaces:

Accessible, Serializable, ImageObserver, MenuContainer

```
public class RoundEdgeContainer
extends JPanel
```

Contains a part of the GUI and surrounds it with a round-edged, colored rectangle.

Constructor Summary	Page
`RoundEdgeContainer(Color color, Component component)` Constructs a new `RoundEdgeContainer`.	124

Method Summary		Page
`protected void`	`paintComponent(Graphics g)` Paints the component.	124

Constructor Detail

RoundEdgeContainer

```
public RoundEdgeContainer(Color color,
                          Component component)
```

Constructs a new `RoundEdgeContainer`.

Parameters:

`color` - The color of the surrounding rectangle.

`component` - The GUI element that should be contained.

Method Detail

paintComponent

```
protected void paintComponent(Graphics g)
```

Paints the component.

Overrides:

`paintComponent` in class `JComponent`

Parameters:

`g` - The Graphics object where to paint the rectangle.

gui.container
UserFunctionContainer

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └gui.container.FunctionContainer
                      └gui.container.UserFunctionContainer
```

All Implemented Interfaces:

ActionListener, EventListener, Accessible, Serializable, ImageObserver, MenuContainer

---

```
public class UserFunctionContainer
extends FunctionContainer
```

Constructor Summary	Page
UserFunctionContainer(GateDrawer parent, KeyListener keyListener) Constructs a new UserFunctionContainer.	125

Method Summary		Page
void	**actionPerformed**(ActionEvent e) Handles action events on the buttons of this container.	125
protected void	**initializeComponent**() Initializations depending of the subtype.	125

Methods inherited from class gui.container.FunctionContainer

getFunction, setButtonsEnabled, setEnabled, setEval, setFontSizes, setFunction

Constructor Detail

UserFunctionContainer

```
public UserFunctionContainer(GateDrawer parent,
                             KeyListener keyListener)
```

Constructs a new UserFunctionContainer.

Parameters:

parent - The parent GUI of this component.

keyListener - A KeyListener for user input.

Method Detail

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

Handles action events on the buttons of this container. Events are passed to the parent GUI.

Specified by:

actionPerformed in interface ActionListener

---

initializeComponent

```
protected void initializeComponent()
```

Description copied from class: FunctionContainer

Initializations depending of the subtype.

Overrides:

`initializeComponent` in class `FunctionContainer`

Package

gui.menubar

Class Summary		Page
CreateExerciseMenuBar	Menubar for creating or editing a set of Exercises.	127
DoExerciseMenuBar	Menubar for executing a set of Exercises.	128
ExerciseMenuBar	A menu bar that provides options to create, execute and edit or displays information about Exercises.	130
ExerciseResultsMenuBar	This menubar displays the results of an Exercise.	131
MenuBar	The menubar of this program.	132

gui.menubar
CreateExerciseMenuBar

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JMenuBar
                  └gui.menubar.ExerciseMenuBar
                      └gui.menubar.CreateExerciseMenuBar
```

All Implemented Interfaces:

ActionListener, EventListener, Accessible, MenuElement, Serializable, ImageObserver, MenuContainer

---

```
public class CreateExerciseMenuBar
extends ExerciseMenuBar
implements ActionListener
```

Menubar for creating or editing a set of Exercises.

Constructor Summary	Page
CreateExerciseMenuBar(GateDrawer gateDrawer, ExerciseService exerciseService) Constructs a new CreateExerciseMenuBar.	127

Method Summary		Page
void	**actionPerformed**(ActionEvent e) Handles ActionEvents from this menubars components.	128
void	**paintComponent**(Graphics g)	128
void	**setToCurrentExercise**() Displays information about the current Exercise such as points and attempts.	128

Constructor Detail

CreateExerciseMenuBar

```
public CreateExerciseMenuBar(GateDrawer gateDrawer,
                             ExerciseService exerciseService)
```

Constructs a new CreateExerciseMenuBar.

Parameters:

gateDrawer - The parent GUI.

exerciseService - The ExerciseService containing the Exercises that are created or edited.

Method Detail

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

Handles ActionEvents from this menubars components.

Specified by:

`actionPerformed` in interface `ActionListener`

Parameters:

`e` - The ActionEvent.

---

setToCurrentExercise

```
public void setToCurrentExercise()
```

Description copied from class: `ExerciseMenuBar`

Displays information about the current `Exercise` such as points and attempts. The current `Exercise` is taken from `ExerciseService`.

Overrides:

`setToCurrentExercise` in class `ExerciseMenuBar`

---

paintComponent

```
public void paintComponent(Graphics g)
```

Overrides:

`paintComponent` in class `JComponent`

gui.menubar
DoExerciseMenuBar

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JMenuBar
                  └gui.menubar.ExerciseMenuBar
                      └gui.menubar.DoExerciseMenuBar
```

All Implemented Interfaces:

ActionListener, EventListener, Accessible, MenuElement, Serializable, ImageObserver, MenuContainer

---

```
public class DoExerciseMenuBar
extends ExerciseMenuBar
implements ActionListener
```

Menubar for executing a set of `Exercise`s.

Constructor Summary	Page
DoExerciseMenuBar(GateDrawer gateDrawer, ExerciseService exerciseService) Constructs a new `DoExerciseMenuBar`.	129

Method Summary		Page
void	**actionPerformed**(ActionEvent e) Handles action events on the menu bar.	129
void	**setToCurrentExercise**() Displays information about the current `Exercise` such as points and attempts.	129

Constructor Detail

DoExerciseMenuBar

```
public DoExerciseMenuBar(GateDrawer gateDrawer,
                         ExerciseService exerciseService)
```

Constructs a new `DoExerciseMenuBar`.

Parameters:

`gateDrawer` - The parent GUI.

`exerciseService` - The service containing the `Exercise`s.

Method Detail

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

Handles action events on the menu bar. All actions are performed in `GateDrawer` or `ExerciseService`.

Specified by:

`actionPerformed` in interface `ActionListener`

---

setToCurrentExercise

`public void` **`setToCurrentExercise`**`()`

Description copied from class: `ExerciseMenuBar`

Displays information about the current `Exercise` such as points and attempts. The current `Exercise` is taken from `ExerciseService`.

Overrides:

`setToCurrentExercise` in class `ExerciseMenuBar`

gui.menubar
ExerciseMenuBar

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JMenuBar
                  └gui.menubar.ExerciseMenuBar
```

All Implemented Interfaces:

Accessible, MenuElement, Serializable, ImageObserver, MenuContainer

Direct Known Subclasses:

CreateExerciseMenuBar, DoExerciseMenuBar, ExerciseResultsMenuBar

---

```
public abstract class ExerciseMenuBar
extends JMenuBar
```

A menu bar that provides options to create, execute and edit or displays information about `Exercise`s.

Constructor Summary	Page
ExerciseMenuBar() Constructs a new `ExerciseMenuBar`.	131

Method Summary		Page
abstract void	**setToCurrentExercise**() Displays information about the current `Exercise` such as points and attempts.	131

Constructor Detail

ExerciseMenuBar

```
public ExerciseMenuBar()
```

Constructs a new `ExerciseMenuBar`.

Method Detail

setToCurrentExercise

```
public abstract void setToCurrentExercise()
```

Displays information about the current `Exercise` such as points and attempts. The current `Exercise` is taken from `ExerciseService`.

gui.menubar
ExerciseResultsMenuBar

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JMenuBar
                  └gui.menubar.ExerciseMenuBar
                      └gui.menubar.ExerciseResultsMenuBar
```

All Implemented Interfaces:

ActionListener, EventListener, Accessible, MenuElement, Serializable, ImageObserver, MenuContainer

---

```
public class ExerciseResultsMenuBar
extends ExerciseMenuBar
implements ActionListener
```

This menubar displays the results of an `Exercise`.

Constructor Summary	Page
ExerciseResultsMenuBar(GateDrawer gui, ExerciseService exerciseService) Constructs a new `ExerciseResultsMenuBar`.	132

Method Summary		Page
void	**actionPerformed**(ActionEvent e) Handles ActionEvents from this menubars components.	132
void	**setToCurrentExercise**() Updates the menubar to display information about the current `Exercise`.	132

Constructor Detail

ExerciseResultsMenuBar

```
public ExerciseResultsMenuBar(GateDrawer gui, ExerciseService exerciseService)
```

Constructs a new `ExerciseResultsMenuBar`. All containing components are initialized and information about the first `Exercise` is displayed.

Parameters:

`gui` - The parent GUI.

`exerciseService` - The `ExerciseService` containing information about the `Exercises` to display.

Method Detail

setToCurrentExercise

```
public void setToCurrentExercise()
```

Updates the menubar to display information about the current `Exercise`.

Overrides:

`setToCurrentExercise` in class `ExerciseMenuBar`

---

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

Handles ActionEvents from this menubars components.

Specified by:

`actionPerformed` in interface `ActionListener`

gui.menubar
MenuBar

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JMenuBar
                  └gui.menubar.MenuBar
```

All Implemented Interfaces:

ActionListener, EventListener, Accessible, MenuElement, Serializable, ImageObserver, MenuContainer

---

```
public class MenuBar
extends JMenuBar
implements ActionListener
```

The menubar of this program. Contains general functions such as loading, saving and info.

Constructor Summary	Page
MenuBar(GateDrawer gateDrawer) Constructs a new Menubar for the program.	133

Method Summary		Page
void	**actionPerformed**(ActionEvent e) Handles action events when a menu item is selected.	133

Constructor Detail

MenuBar

```
public MenuBar(GateDrawer gateDrawer)
```

Constructs a new Menubar for the program.

Parameters:

`gateDrawer` - The main GUI of the program.

Method Detail

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

Handles action events when a menu item is selected. All actions are performed by the parent GUI.

Specified by:

`actionPerformed` in interface `ActionListener`

Package

gui.popup

Class Summary		Page
ConnectionPopup	Shows a popup in the DrawingArea with options to edit a GateConnection.	134
GatePopup	Shows a popup in DrawingArea to edit a gate.	134
Popup	Shows a popup in DrawingArea to edit a part of it.	135
SelectionPopup	Shows a popup in DrawingArea to edit a selection.	137

gui.popup
ConnectionPopup

```
java.lang.Object
  └ gui.popup.Popup
      └ gui.popup.ConnectionPopup
```

All Implemented Interfaces:

ActionListener, EventListener

```
public class ConnectionPopup
extends Popup
```

Shows a popup in the DrawingArea with options to edit a GateConnection.

Constructor Summary	Page
ConnectionPopup(GateConnection connection, Controller controller) Constructs a new ConnectionPopup.	134

Method Summary		Page
void	**actionPerformed**(ActionEvent e) Edits the GateConnection after the user has clicked on an option.	134

Constructor Detail

ConnectionPopup

```
public ConnectionPopup(GateConnection connection,
                       Controller controller)
```

Constructs a new ConnectionPopup.

Parameters:

connection - The connection to edit.

controller - The controller of this program.

Method Detail

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

Edits the GateConnection after the user has clicked on an option. The popup is set invisible.

Specified by:

actionPerformed in interface ActionListener

gui.popup
GatePopup

```
java.lang.Object
  └ gui.popup.Popup
      └ gui.popup.GatePopup
```

All Implemented Interfaces:

ActionListener, EventListener

```
public class GatePopup
extends Popup
```

Shows a popup in `DrawingArea` to edit a gate.

Constructor Summary	Page
GatePopup(DockingPoint owner, Controller controller) Constructs a new `GatePopup`.	135

Method Summary		Page
void	**actionPerformed**(ActionEvent e) Edits the gate after user has clicked on an option.	135

Constructor Detail

GatePopup

```
public GatePopup(DockingPoint owner,
                 Controller controller)
```

Constructs a new `GatePopup`.

Parameters:

`owner` - `DockingPoint` that is to be edited. Popup is displayed over this `DockingPoint`.

`controller` - The controler of this program.

Method Detail

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

Edits the gate after user has clicked on an option. The popup is set invisible.

Specified by:

`actionPerformed` in interface `ActionListener`

gui.popup
Popup

```
java.lang.Object
  └ gui.popup.Popup
```

All Implemented Interfaces:

ActionListener, EventListener

Direct Known Subclasses:

ConnectionPopup, GatePopup, SelectionPopup

```
public abstract class Popup
extends Object
implements ActionListener
```

Shows a popup in `DrawingArea` to edit a part of it.

Constructor Summary	Page
Popup(Controller controller) Constructs a new Popup.	136

Method Summary		Page
JPopupMenu	**getPopup**() Returns the actual popup.	137
boolean	**isVisible**() Returns `True` if the popup is visible.	136
void	**setVisible**(boolean visible) Sets the popup visible.	136

Constructor Detail

Popup

```
public Popup(Controller controller)
```

Constructs a new Popup.

Parameters:

`controller` - The controller of this program.

Method Detail

isVisible

```
public boolean isVisible()
```

Returns `True` if the popup is visible. `False` otherwise.

Returns:

`True` if the popup is visible. `False` otherwise.

---

setVisible

```
public void setVisible(boolean visible)
```

Sets the popup visible.

Parameters:

`visible` - `True` if the popup should be visible. `False` if not.

---

getPopup

```
public JPopupMenu getPopup()
```

Returns the actual popup.

Returns:

The actual popup.

gui.popup
SelectionPopup

```
java.lang.Object
  └ gui.popup.Popup
      └ gui.popup.SelectionPopup
```

All Implemented Interfaces:

ActionListener, EventListener

```
public class SelectionPopup
extends Popup
```

Shows a popup in `DrawingArea` to edit a selection.

Constructor Summary	Page
SelectionPopup(Controller controller) Constructs a new SelectionPopup.	138

Method Summary		Page
void	**actionPerformed**(ActionEvent e) Edits the selection if the user chose an option.	138

Constructor Detail

SelectionPopup

```
public SelectionPopup(Controller controller)
```

Constructs a new `SelectionPopup`.

Parameters:

`controller` - The controller of this program.

Method Detail

actionPerformed

```
public void actionPerformed(ActionEvent e)
```

Edits the selection if the user chose an option. The popup is set invisible.

Specified by:

`actionPerformed` in interface `ActionListener`

Package

service

Class Summary		Page
ConnectionService	This service class handles all calculation concerning lines in the drawingArea.	139
Controller	The controller class is the main controller for the whole project.	140
DrawingService	This service class handles everything connected with the drawing of logic circuits, represented by Graphs.	146
ExerciseService	This service manages all operations concerning Exercises.	153
FunctionService	This service is used to generate random boolean functions.	160
GridService	This service manages the grid displayed in the DrawingArea.	161
ImageService	Manages all images used by the program.	165
LogicService	Service managing all boolean operations in the program.	172
QualifyerService	Service that manages Qualifyers for the program.	175

service

ConnectionService

```
java.lang.Object
  └ service.ConnectionService
```

public class **ConnectionService**
extends Object

This service class handles all calculation concerning lines in the drawingArea. Calculations are necessary when gates or lines are moved, especially when the current node in drawingService is moved.

Constructor Summary	Page
ConnectionService(int gateHeight, GridService grid) Constructs a new ConnectionService with all necessary parameters for calculations.	139

Method Summary		Page
void	**dragCurrentNode**(Node node, int mouseX, int mouseY, int oldMouseX, int oldMouseY) Drags the line of a given Node from old mouseposition to new mouseposition.	140
void	**mergeLines**(ArrayList<Line2D.Double> lines) Checks a list of lines if two consecutive lines are on the same level (have the same x-coordinates) and merges them if so.	140
void	**resizeConnectionEnd**(GateConnection connection, Point end) Moves the end of a GateConnection to the new Point end.	140
void	**resizeConnectionStart**(GateConnection connection, Point start) Moves the start of a GateConnection to the new Point start.	140

Constructor Detail

ConnectionService

public **ConnectionService**(int gateHeight,
GridService grid)

Constructs a new ConnectionService with all necessary parameters for calculations.

Parameters:

gateHeight - The Height of a gate in the DrawingArea.

grid - The GridService object handling the grid that is currently used by the DrawingService.

Method Detail

mergeLines

```
public void mergeLines(ArrayList<Line2D.Double> lines)
```

Checks a list of lines if two consecutive lines are on the same level (have the same x-coordinates) and merges them if so.

Parameters:

`lines` - A list of lines to check.

---

resizeConnectionEnd

```
public void resizeConnectionEnd(GateConnection connection,
                                Point end)
```

Moves the end of a `GateConnection to the new Point end. The connection is reshaped and broken into additional lines if necessary.`

Parameters:

`connection` - The `GateConnection to resize.`

`end` - The new end point of the connection.

---

resizeConnectionStart

```
public void resizeConnectionStart(GateConnection connection,
                                  Point start)
```

Moves the start of a `GateConnection to the new Point start. The connection is reshaped and broken into additional lines if necessary.`

Parameters:

`connection` - The `GateConnection to resize.`

`start` - The new start point of the connection.

---

dragCurrentNode

```
public void dragCurrentNode(Node node,
                            int mouseX,
                            int mouseY,
                            int oldMouseX,
                            int oldMouseY)
```

Drags the line of a given `Node` from old mouseposition to new mouseposition. Important: the node holds the line that is dragged. If this very line is deleted during dragging, the node is updated with a new line.

Parameters:

`node` - Node that is dragged by the user.

`mouseX` - New x-position of the mouse.

`mouseY` - New y-position of the mouse.

`oldMouseX` - Old x-position of the mouse.

`oldMouseY` - Old y-position of the mouse.

service
Controller

```
java.lang.Object
  └ service.Controller
```

```
public class Controller
extends Object
```

The controller class is the main controller for the whole project. It instantiates the GUI and most of the service classes. Further more the controller handles GUI requests for services.

Constructor Summary	Page
Controller() Constructs and initializes a new controller.	142

Method Summary		Page
void	**abortCurrentLine**() Aborts drawing of the current line.	142
void	**clear**() Clears the current Graph.	143
boolean	**compareFunctionAndGraph**() Compares the current user-entered function and graph (from drawingService).	143
void	**deleteComponent**(DrawableComponent component) Handles a user request for deletion of a component in drawingArea.	145
void	**deleteConnection**(GateConnection connection) Handles a user request for deletion of a line in drawingArea.	145
void	**deleteSelection**() Handles user request of deleting the current selection.	144
boolean	**evaluateFunction**() Returns the logical value of the user-entered boolean function.	144
boolean	**evaluateGraph**() Returns the logical value of the current graph.	144
void	**export**(File file) Exports the current graph as an image to a given file.	144
void	**functionToGraph**(String function) Draws a boolean function as a graph.	144
void	**functionToTree**() Creates a boolean tree out of the user-entered function.	143
void	**generateGraph**() Randomly generates a new graph and draws it in drawingArea.	144
Graph	**getGraph**() Returns the current graph.	145
String	**getGraphAsFunction**() Returns the boolean function that corresponds the current graph.	144
Graph	**getGraphCopy**() Returns a deep copy of the current graph.	145
String	**getNewFunction**() Returns a new random boolean function.	143
boolean	**isFunctionSatisfiable**(String function) Tests if a given function is satisfiable.	145
boolean	**isGraphSatisfiable**() Tests if the current graph is satisfiable.	146
void	**load**(File file) Loads graph and function from a given file.	142
void	**mouseClicked**(MouseEvent event) Handles a click in the drawingArea.	143

void	**objectClicked**(DrawableComponent g, Point mousePosition) Handles a click on a drawableComponent.	143
void	**randomValues**() Creates a random assignment for all operands.	145
void	**repaint**() Passes a repaint request to the GUI.	145
void	**save**(File file) Saves the current state (graph + user-entered function) to a given file.	142
void	**setCurrentMode**(ButtonQualifyer qualifyer) Setter for the current mode.	143
void	**setGraph**(Graph graph) Sets the current graph in drawingService to a deep copy of the given graph.	145

Constructor Detail

Controller

```
public Controller()
```

Constructs and initializes a new `controller`. A new GUI (`gateDrawer`) is instantiated as well as a `drawingService` and a `logicService`.

Method Detail

load

```
public void load(File file)
    throws Exception
```

Loads graph and function from a given file. Fileaccess is done in fileDAO.

Parameters:

`file` - File to load from.

Throws:

`Exception` - If the file is not found or content cannot be loaded from it.

---

save

```
public void save(File file)
    throws Exception
```

Saves the current state (graph + user-entered function) to a given file. Fileaccess is done in fileDAO.

Parameters:

`file` - File to save the state to.

Throws:

`Exception` - If a problem with the file occurs.

---

abortCurrentLine

```
public void abortCurrentLine()
```

Aborts drawing of the current line. Action is executed in the drawingService.

---

objectClicked

```
public void objectClicked(DrawableComponent g,
                          Point mousePosition)
```

Handles a click on a drawableComponent. Decides to draw a line or to remove a gate according to the current mode.

Parameters:

`g` - DrawableComponent that was clicked by the user.

`mousePosition` - Position on the drawingArea of the click.

---

mouseClicked

```
public void mouseClicked(MouseEvent event)
```

Handles a click in the drawingArea. Decides to draw a line or to create a new gate or operand in the drawingArea according to the current mode.

Parameters:

`event` - ClickEvent needed for the coordinates of the mouse click.

---

setCurrentMode

```
public void setCurrentMode(ButtonQualifyer qualifyer)
```

Setter for the current mode.

Parameters:

`qualifyer` - Qualifyer defining the current mode.

---

compareFunctionAndGraph

```
public boolean compareFunctionAndGraph()
```

Compares the current user-entered function and graph (from drawingService). Action is executed in the logicService.

Returns:

- True if function and graph are equal.
- False if not.

---

getNewFunction

```
public String getNewFunction()
```

Returns a new random boolean function. Action is executed in functionService.

Returns:

A new randomly generated function

---

clear

```
public void clear()
```

Clears the current Graph. Action is executed in drawingService.

---

functionToTree

```
public void functionToTree()
```

Creates a boolean tree out of the user-entered function. Action is executed in logicService.

---

getGraphAsFunction

`public String` **`getGraphAsFunction`**`()`

Returns the boolean function that corresponds the current graph. Action is executed in logicService.

Returns:

Function corresponding the current graph.

---

evaluateFunction

`public boolean` **`evaluateFunction`**`()`

Returns the logical value of the user-entered boolean function. Action is executed in logicService.

Returns:

Logical value of the user-entered function.

---

evaluateGraph

`public boolean` **`evaluateGraph`**`()`

Returns the logical value of the current graph. Action is executed in logicService.

Returns:

Logical value of the current graph.

---

generateGraph

`public void` **`generateGraph`**`()`

Randomly generates a new graph and draws it in drawingArea. Random generation is done in functionService. Drawing is done in drawingService.

---

functionToGraph

`public void` **`functionToGraph`**`(String function)`

Draws a boolean function as a graph. Drawing is done in method drawGraph and in drawingService.

Parameters:

`function` - The function that should be drawn.

---

export

`public void` **`export`**`(File file)`
`throws Exception`

Exports the current graph as an image to a given file. Assembling of the image is done in drawingService. File is saved in FileDAO.

Parameters:

`file` - File to save the image to.

Throws:

`Exception` - If an error with the file occurs.

---

deleteSelection

`public void` **`deleteSelection`**`()`

Handles user request of deleting the current selection. Deletion is done in drawingService.

---

repaint

`public void` **`repaint`**`()`

Passes a repaint request to the GUI.

---

deleteConnection

`public void` **`deleteConnection`**`(GateConnection connection)`

Handles a user request for deletion of a line in drawingArea. Request is executed by drawingService.

Parameters:

`connection` - Connection to delete.

---

deleteComponent

`public void` **`deleteComponent`**`(DrawableComponent component)`

Handles a user request for deletion of a component in drawingArea. Request is executed by drawingService.

Parameters:

`component` - Component to delete.

---

getGraphCopy

`public Graph` **`getGraphCopy`**`()`

Returns a deep copy of the current graph. Deep copy is created in the graph itself.

Returns:

A new deep copy of the current graph.

---

getGraph

`public Graph` **`getGraph`**`()`

Returns the current graph. Changes made on this `Graph` are visible in the `DrawingArea`!

Returns:

The current graph.

---

setGraph

`public void` **`setGraph`**`(Graph graph)`

Sets the current graph in drawingService to a deep copy of the given graph.

Parameters:

`graph` - Graph to set.

---

randomValues

`public void` **`randomValues`**`()`

Creates a random assignment for all operands.

---

isFunctionSatisfiable

`public boolean` **`isFunctionSatisfiable`**`(String function)`

Tests if a given function is satisfiable. Testing is done in logicService.

Parameters:

`function` - The Boolean Function that has to be tested.

Returns:

`True` if the function is satisfiable. `False` otherwise.

---

isGraphSatisfiable

`public boolean` **`isGraphSatisfiable`**`()`

Tests if the current graph is satisfiable. Testing is done in logicService.

Returns:

`True` if the `Graph` is satisfiable. `False` otherwise.

service
DrawingService

```
java.lang.Object
  └ service.DrawingService
```

All Implemented Interfaces:

ComponentListener, EventListener

```
public class DrawingService
extends Object
implements ComponentListener
```

This service class handles everything connected with the drawing of logic circuits, represented by `Graph`s. Some specific services are delegated to other services: handling of `GateConnections` (`ConnectionService`) and handling of the grid (`GridService`). The service acts as a `ComponentListener` for the drawingArea and therefore handles the resizing of the grid.

Constructor Summary	Page
DrawingService(GateDrawer gui, Controller controller) Constructs a new `DrawingService`.	148

Method Summary		Page
void	**abortCurrentLine**() Aborts drawing of the current `GateConnection`.	149
DrawableComponent	**addComponent**(BooleanQualifyer qualifyer, int x, int y) Adds a new `DrawableComponent` to the `DrawingArea`.	149
void	**clear**() Removes all `Gate`s and `GateConnection`s from the `DrawingArea`.	150
void	**componentHidden**(ComponentEvent event) Component hidden event is ignored.	153
void	**componentMoved**(ComponentEvent event) Component moved event is ignored.	153
void	**componentResized**(ComponentEvent event) Handles component resized event.	153
void	**componentShown**(ComponentEvent event) Handles component shown event.	153
void	**deleteCurrentNodeLine**() Deletes the `GateConnection` contained by the `currentNode`.	152
void	**deleteSelection**() Deletes the current selection.	149
void	**dragCurrentComponent**() Moves the `currentComponent` when user dragged it.	150
void	**drawConnection**(DockingPoint out, DockingPoint in) Creates a GateConnection between the specified `DockingPoint`s.	151
int	**drawGate**(QualifyerGate qg, int leftID, int rightID, boolean negated, boolean negatedLeft, boolean negatedRight) Draws a `Gate` and its Connections automatically.	150
void	**drawLine**(DrawableComponent obj, Point mousePosition) Draws a `GateConnection` between two `DrawableComponent`s.	149
int	**drawOperand**(QualifyerOperand qualifyer) Draws an `Operand, if it is not already drawn.`	150
BufferedImage	**export**() Assembles an image from the `Graph` for export as an image.	151
void	**flush**() Removes all `Gate`s and `GateConnection`s from the `DrawingArea`.	150

ArrayList<GateConnection>	**getConnections**() Getter for GateConnections currently displayed in the DrawingArea.	151
DrawableComponent	**getCurrentComponent**() Returns the currentComponent.	152
ArrayList<DrawableComponent>	**getDrawables**() Getter for DrawableComponents currently displayed in the DrawingArea.	151
Graph	**getGraph**() Returns the current Graph.	152
Graph	**getGraphCopy**() Returns a deep copy of the Graph currently displayed in DrawingArea.	152
int	**getInset**() Returns the insets of the DrawingArea.	152
boolean	**isDragged**() Returns if something is currently dragged.	153
boolean	**isSelected**() Returns if the user has selected something in the DrawingArea.	152
boolean	**isSelecting**() Returns if the user is currently selecting in the DrawingArea.	152
void	**setDragged**(boolean b) Sets dragged to b.	149
void	**setGraph**(Graph graph) Sets the currently displayed Graph to the passed Graph.	151
void	**setNegation**(int id, boolean negation) Sets the negation of the Out-DockingPoint of a DrawableComponent.	151
void	**setSelected**(boolean b) Sets selected.	152
void	**setSelecting**(boolean b) Sets selecting.	153
void	**updateConnection**(GateConnection connection, Point start, Point end) Updates the passed connection.	149

Constructor Detail

DrawingService

```
public DrawingService(GateDrawer gui,
                      Controller controller)
```

Constructs a new DrawingService. A new, empty Graph is instantiated and the listeners for the DrawingArea are set.

Parameters:

gui - The GUI for this application. Needed to get the DrawingArea.

controller - The main Controller for this program, acting as interface between GUI and service.

Method Detail

addComponent

```
public DrawableComponent addComponent(BooleanQualifyer qualifyer,
                                      int x,
                                      int y)
```

Adds a new `DrawableComponent` to the `DrawingArea`.

Parameters:

`qualifyer` - The `Qualifyer` for the new `DrawableComponent`.

`x` - The x coordinate of the `DrawableComponent`.

`y` - The y coordinate of the `DrawableComponent`.

Returns:

The new `DrawableComponent`.

---

drawLine

```
public void drawLine(DrawableComponent obj,
                     Point mousePosition)
```

Draws a `GateConnection` between two `DrawableComponent`s. If no line if currently being drawn, a new line, starting at the `DockingPoint` at `mousePosition` is drawn. If a line is already being drawn, its end point is set to the `DockingPoint` at `mousePosition`.

Parameters:

`obj` - the `DrawableComponent` containing the `DockingPoint` that is used to draw the line.

`mousePosition` - The position of the mouse click to find the correct `DockingPoint`.

---

updateConnection

```
public void updateConnection(GateConnection connection,
                             Point start,
                             Point end)
```

Updates the passed connection. The new end of the line is at the current mouse position.

Parameters:

`connection` - The `GateConnection` that should be updated.

`start` - The start point of the line.

`end` - The new end point of the line.

---

abortCurrentLine

```
public void abortCurrentLine()
```

Aborts drawing of the current `GateConnection`.

---

deleteSelection

```
public void deleteSelection()
```

Deletes the current selection.

---

setDragged

```
public void setDragged(boolean b)
```

Sets `dragged` to `b`.

Parameters:

`b` - `True` if the user currently drags. `False` otherwise.

---

dragCurrentComponent

```
public void dragCurrentComponent()
```

Moves the `currentComponent` when user dragged it.

---

clear

```
public void clear()
```

Removes all `Gate`s and `GateConnection`s from the `DrawingArea`. The `Graph` gets cleared without instantiating a new one. All fields indicating a user action, such as `selecting` are set to false.

---

flush

```
public void flush()
```

Removes all `Gate`s and `GateConnection`s from the `DrawingArea`. The `Graph` is reinstantiated, not just cleared. All fields indicating a user action, such as `selecting` are set to false.

---

drawOperand

```
public int drawOperand(QualifyerOperand qualifyer)
```

Draws an `Operand, if it is not already drawn.`

Parameters:

`qualifyer` - `QualifyerOperand` specifing the `Operand`.

Returns:

The index of the `Operand`.

---

drawGate

```
public int drawGate(QualifyerGate qg,
                    int leftID,
                    int rightID,
                    boolean negated,
                    boolean negatedLeft,
                    boolean negatedRight)
```

Draws a `Gate` and its Connections automatically. The column of the gate is calculated by leftID- and rightID-column.

Parameters:

`qg` - The qualifyer specifying what `Gate` to draw.

`leftID` - Id of the `Gate` connected at the left `DockingPoint`.

`rightID` - If of the `Gate` connected at the right `DockingPoint`.

`negated` - True if the Out-`DockingPoint` is negated. False if not.

`negatedLeft` - True if the In1-`DockingPoint` is negated. False if not.

`negatedRight` - True if the In1-`DockingPoint` is negated. False if not.

Returns:

The id of the created `Gate`.

---

setNegation

```
public void setNegation(int id,
                        boolean negation)
```

Sets the negation of the Out-`DockingPoint` of a `DrawableComponent`.

Parameters:

`id` - The id of the `DrawableComponent`.

`negation` - True if the `DockingPoint` should be negated. False if not.

---

drawConnection

```
public void drawConnection(DockingPoint out,
                           DockingPoint in)
```

Creates a GateConnection between the specified `DockingPoint`s.

Parameters:

`out` - The start `DockingPoint`.

`in` - The end `DockingPoint`.

---

export

```
public BufferedImage export()
```

Assembles an image from the `Graph` for export as an image.

Returns:

The current graph as an image.

---

getDrawables

```
public ArrayList<DrawableComponent> getDrawables()
```

Getter for `DrawableComponent`s currently displayed in the `DrawingArea`. Used for modifications of the current display.

Returns:

`ArrayList` of currently displayed `DrawableComponent`s.

---

getConnections

```
public ArrayList<GateConnection> getConnections()
```

Getter for `GateConnections` currently displayed in the `DrawingArea`. Used for modifications of the current display.

Returns:

`ArrayList` of currently displayed `GateConnection`s.

---

setGraph

```
public void setGraph(Graph graph)
```

Sets the currently displayed `Graph` to the passed `Graph`. A deep copy will be made so changes on the original `Graph` will not be displayed

Parameters:

`graph` - `Graph` to display in the `DrawingArea`.

---

deleteCurrentNodeLine

`public void` **`deleteCurrentNodeLine`**`()`

Deletes the `GateConnection` contained by the `currentNode`.

---

getGraph

`public` `Graph` **`getGraph`**`()`

Returns the current `Graph`.

Returns:

The current `Graph`.

---

getGraphCopy

`public` `Graph` **`getGraphCopy`**`()`

Returns a deep copy of the `Graph` currently displayed in `DrawingArea`. Changes in the returned `Graph` will not be displayed.

Returns:

A deep copy of the `Graph` currently displayed.

---

getInset

`public int` **`getInset`**`()`

Returns the insets of the `DrawingArea`.

Returns:

The insets of the `DrawingArea`.

---

isSelecting

`public boolean` **`isSelecting`**`()`

Returns if the user is currently selecting in the `DrawingArea`.

Returns:

True if the user currently selects; False otherwise.

---

isSelected

`public boolean` **`isSelected`**`()`

Returns if the user has selected something in the `DrawingArea`.

Returns:

`True` if something is currently selected. `False` otherwise.

---

getCurrentComponent

`public` `DrawableComponent` **`getCurrentComponent`**`()`

Returns the `currentComponent`.

Returns:

The `currentComponent`.

---

setSelected

`public void` **`setSelected`**`(boolean b)`

Sets `selected`.

Parameters:

`b` - `True` if the user has selected something. `False` otherwise.

---

setSelecting

`public void` **`setSelecting`**`(boolean b)`

Sets `selecting`.

Parameters:

`b` - `True` if the user is selecting something. `False` otherwise.

---

isDragged

`public boolean` **`isDragged`**`()`

Returns if something is currently dragged.

Returns:

`True` if something is currently dragged. `False otherwise`.

---

componentHidden

`public void` **`componentHidden`**`(ComponentEvent event)`

Component hidden event is ignored.

Specified by:

`componentHidden` in interface `ComponentListener`

---

componentMoved

`public void` **`componentMoved`**`(ComponentEvent event)`

Component moved event is ignored.

Specified by:

`componentMoved` in interface `ComponentListener`

---

componentResized

`public void` **`componentResized`**`(ComponentEvent event)`

Handles component resized event. Calls resize() with the new width and height of the `DrawingArea`.

Specified by:

`componentResized` in interface `ComponentListener`

---

componentShown

`public void` **`componentShown`**`(ComponentEvent event)`

Handles component shown event. Sets focus to `DrawingArea`.

Specified by:

`componentShown` in interface `ComponentListener`

service

ExerciseService

```
java.lang.Object
  └ service.ExerciseService
```

```
public class ExerciseService
extends Object
```

This service manages all operations concerning `Exercise`s.

Constructor Summary	Page
ExerciseService() Constructs a new `ExerciseService`.	155
ExerciseService(ArrayList<Exercise> exercises) Constructs a new `ExerciseService` with an existing collection of `Exericse`s.	155

Method Summary		Page
void	**deleteCurrentExercise**() Deletes the current `Exercise`.	155
String	**evaluateCurrentExercise**(Graph graph, String function) Evaluates the current `Exercise`.	158
void	**finish**(File file) Saves the collection of `Exercise`s, that is managed by this service, to a given file.	159
void	**finishExecution**() Saves the collection of `Exercise`s, that is managed by this class, to its original file.	159
Exercise	**getCurrentExercise**() Returns `Exercise` at `currentExercise` position.	157
int	**getCurrentExerciseNumber**() Returns the number of the `Exercise` currently displayed.	157
ArrayList<Exercise>	**getExercises**() Returns the collection of `Exercise`s managed by this service.	157
Exercise	**getNextExercise**() Returns the next `Exercise` after `currentExercise` position.	157
int	**getNumberSolved**() Returns the number of solved `Exercise`s.	160
int	**getNumberTried**() Returns the number of tried `Exericse`s.	160
int	**getPointsReached**() Returns the sum of all points that were reached yet.	157
int	**getPointsTotal**() Returns the sum of all points that can be reached in total.	158
Exercise	**getPreviousExercise**() Returns the `Exercise` before `currentExercise` position.	157
int	**getTotalExerciseNumber**() Returns the total number of `Exercise`s managed by this service.	157
void	**giveUpCurrentExercise**(GateDrawer gui, Controller controller) Gives up the current `Exercise`.	159
void	**load**(File file) Loads a collection of `Exercise`s from a given file.	158
void	**loadForExecution**(File file) Loads a collection of `Exercise`s from a file for execution.	160
void	**loadForResults**(File file) Loads a collection of `Exercise`s from a file to display the results.	160

void	**newExercise**() Inserts a new empty Exercise add the end of exercises.	155
void	**next**() Moves to the Exercise at currentPosition + 1.	158
void	**previous**() Moves to the Exercise at currentPosition - 1.	158
void	**saveExercise**(ExerciseType mode, Graph graph, String userFunction, int points, int attempts) Creates and saves a new Exercise at the currentExercise position.	159
void	**saveExercise**(Graph graph, int points, int attempts) Creates and saves a new FunctionExercise with given graph, points and attempts.	156
void	**saveExercise**(String function, int points, int attempts) Creates and saves a new GraphExercise with given function, points and attempts.	156
void	**saveIncorrectFunctionExercise**(String function, Graph graph, int points, int attempts) Creates and saves a new IncorrectFunctionExercise with given graph, function, points and attempts.	156
void	**saveIncorrectGraphExercise**(String function, Graph graph, int points, int attempts) Creates and saves a new IncorrectGraphExercise with given graph, function, points and attempts.	156

Constructor Detail

ExerciseService

```
public ExerciseService()
```

Constructs a new ExerciseService. exercises is initialized and editable is set to true.

ExerciseService

```
public ExerciseService(ArrayList<Exercise> exercises)
```

Constructs a new ExerciseService with an existing collection of Exericses.

Parameters:

exercises - A collection of Exercises that should be managed by this service.

Method Detail

deleteCurrentExercise

```
public void deleteCurrentExercise()
```

Deletes the current Exercise. If no Exercise exists, no changes are made.

newExercise

```
public void newExercise()
```

Inserts a new empty Exercise add the end of exercises. currentExercise is set to exercises.size() - 1. If the last Exercise is empty, no changes are made.

saveExercise

```
public void saveExercise(String function,
                         int points,
                         int attempts)
```

Creates and saves a new `GraphExercise` with given function, points and attempts. See `save(Exercise exercise)` for details on saving.

Parameters:

`function` - The function of the `GraphExercise`.

`points` - Points the `Exercise` is worth.

`attempts` - Attempts a user may try this `Exercise`.

---

saveExercise

```
public void saveExercise(Graph graph,
                         int points,
                         int attempts)
```

Creates and saves a new `FunctionExercise` with given graph, points and attempts. See `save(Exercise exercise)` for details on saving.

Parameters:

`graph` - The graph of the `FunctionExercise`.

`points` - Points the `Exercise` is worth.

`attempts` - Attempts a user may try this `Exercise`.

---

saveIncorrectFunctionExercise

```
public void saveIncorrectFunctionExercise(String function,
                                          Graph graph,
                                          int points,
                                          int attempts)
```

Creates and saves a new `IncorrectFunctionExercise` with given graph, function, points and attempts. See `save(Exercise exercise)` for details on saving.

Parameters:

`function` - The function of the `IncorrectFunctionExercise`.

`graph` - The graph of the `IncorrectFunctionExercise`.

`points` - Points the `Exercise` is worth.

`attempts` - Attempts a user may try this `Exercise`.

---

saveIncorrectGraphExercise

```
public void saveIncorrectGraphExercise(String function,
                                       Graph graph,
                                       int points,
                                       int attempts)
```

Creates and saves a new `IncorrectGraphExercise` with given graph, function, points and attempts. See `save(Exercise exercise)` for details on saving.

Parameters:

`function` - The function of the `IncorrectGraphExercise`.

`graph` - The graph of the `IncorrectGraphExercise`.

`points` - Points the `Exercise` is worth.

`attempts` - Attempts a user may try this `Exercise`.

---

getCurrentExercise

`public Exercise getCurrentExercise()`

Returns `Exercise` at `currentExercise` position.

Returns:

`Exercise` at `currentExercise` position or `null` if not existent.

---

getNextExercise

`public Exercise getNextExercise()`

Returns the next `Exercise` after `currentExercise` position.

Returns:

Next `Exercise` after `currentExercise` position or `null` of not existent.

---

getPreviousExercise

`public Exercise getPreviousExercise()`

Returns the `Exercise` before `currentExercise` position.

Returns:

`Exercise` before `currentExercise` position or `null` of not existent.

---

getTotalExerciseNumber

`public int getTotalExerciseNumber()`

Returns the total number of `Exercise`s managed by this service.

Returns:

Total number of `Exercises or 0 if no Exercise is initialized.`

---

getCurrentExerciseNumber

`public int getCurrentExerciseNumber()`

Returns the number of the `Exercise` currently displayed.

Returns:

Number of the current exercise or 0 if `exercises` is not initialized.

---

getExercises

`public ArrayList<Exercise> getExercises()`

Returns the collection of `Exercise`s managed by this service.

Returns:

Collection of `Exercise`s managed by this service.

---

getPointsReached

`public int getPointsReached()`

Returns the sum of all points that were reached yet.

Returns:

Sum of all reached points.

---

getPointsTotal

```
public int getPointsTotal()
```

Returns the sum of all points that can be reached in total.

Returns:

Sum of all reachable points.

---

evaluateCurrentExercise

```
public String evaluateCurrentExercise(Graph graph,
                                      String function)
```

Evaluates the current `Exercise`. The correct solution for an `exercise` is calculated and compared to the user's solution. If the solution is correct, the current `Exercise` is set to solved. The solution is saved in the `Exercise`. Whether the solution is correct or not, the number of attempts is raised by one.

Evaluation is only done, if the number of allowed attempts is not yet exceeded!

Parameters:

`graph` - Graph that is currently displayed and might be part of the solution, depending of the type of `Exercise`.

`function` - Function that is currently displayed and might be part o the solution, depending of the type of `Exercise`.

Returns:

A `String` to display according to the result of the evaluation:

- "Die Aufgabe wurde bereits aufgegeben!"
- "Die Aufgabe wurde bereits erledigt!"
- "Es wurden bereits so viele Versuche für diese Aufgabe benötigt, wie möglich sind!"
- "Die Lösung ist korrekt!"
- "Die Lösung ist nicht korrekt!"

---

previous

```
public void previous()
```

Moves to the `Exercise` at `currentPosition` - 1.

---

next

```
public void next()
```

Moves to the `Exercise` at `currentPosition` + 1.

---

load

```
public void load(File file)
    throws Exception
```

Loads a collection of `Exercise`s from a given file.

Parameters:

`file` - File to load `Exercise`s from.

Throws:

`Exception` - If an error occurs while reading the file.

---

saveExercise

```
public void saveExercise(ExerciseType mode,
                         Graph graph,
                         String userFunction,
                         int points,
                         int attempts)
                  throws Exception
```

Creates and saves a new `Exercise` at the `currentExercise` position.

Parameters:

`mode` - The type of the `Exercise`.

`graph` - The logic circuit of the `Exercise`. Can be `null` depending on the type.

`userFunction` - The Boolean function of the `Exercise`. Can be `null` depending on the type.

`points` - The points this `Exercise` is worth.

`attempts` - The attempts that may be made for solving this `Exercise`.

Throws:

`Exception` - If one or more necessary parameters are `null`.

---

finish

```
public void finish(File file)
            throws IllegalArgumentException,
                   Exception
```

Saves the collection of `Exercise`s, that is managed by this service, to a given file. Writing is done in `ExerciseDAO`.

Parameters:

`file` - File to that the `Exercise`s should be saved.

Throws:

`IllegalArgumentException` - If the file is null or invalid.

`Exception` - If an exception occurs while writing the collection to the file.

---

finishExecution

```
public void finishExecution()
                    throws FileNotFoundException,
                           Exception
```

Saves the collection of `Exercise`s, that is managed by this class, to its original file. This method is only called at the end of an execution to store the results in the original file. Writing is done in `ExerciseDAO`.

Throws:

`FileNotFoundException` - If the original file does not exist anymore.

`Exception` - If an exception occurs while writing the collection to the file.

---

giveUpCurrentExercise

```
public void giveUpCurrentExercise(GateDrawer gui,
                                  Controller controller)
```

Gives up the current `Exercise. If the Exercise was not already done, giveUp is set to true in this Exercise. The correct solution of this Exercise is displayed.`

Parameters:

`gui` - The GUI to display the correct solution.

`controller` - The controller to set the `Graph` for the solution.

---

loadForExecution

```
public void loadForExecution(File file)
                      throws Exception
```

Loads a collection of `Exercise`s from a file for execution. Uses `load(File file)`.

Parameters:

`file` - File to load `Exercise`s from.

Throws:

`Exception` - If an error occurs while reading from the file.

---

loadForResults

```
public void loadForResults(File file)
                    throws Exception
```

Loads a collection of `Exercise`s from a file to display the results. Uses/ `load(File file)`.

Parameters:

`file` - File to load `Exercise`s from.

Throws:

`Exception` - If an error occurs while reading from the file.

---

getNumberSolved

```
public int getNumberSolved()
```

Returns the number of solved `Exercise`s.

Returns:

Number of solved `Exercise`s.

---

getNumberTried

```
public int getNumberTried()
```

Returns the number of tried `Exericse`s.

Returns:

Number of tried `Exercise`s.

FunctionService

```
java.lang.Object
  └ service.FunctionService
```

```
public class FunctionService
extends Object
```

This service is used to generate random boolean functions. This class is implemented as a singleton.

Method Summary		Page
static FunctionService	**getInstance**() Getter for the singular instance of this class.	161
String	**getNewFunction**() Generates a new random function that has not yet been generated.	161

Method Detail

getInstance

```
public static FunctionService getInstance()
```

Getter for the singular instance of this class.

Returns:

The singular instance of this class.

getNewFunction

```
public String getNewFunction()
```

Generates a new random function that has not yet been generated. The generated function can be composed of 1 to 5 terms, each containing 1 or 2 operands and 0 or 1 operators. Two terms are connected by an operator. The chances for functions with x terms are:

- 1 term: 20%
- 2 terms: 50%
- 3 terms: 20%
- 4 terms: 6%
- 5 terms: 4%

Returned functions are unique. They have not been generated yet and will never again be returned.

Returns:

Returns a new random function.

service
GridService

```
java.lang.Object
  └ service.GridService
```

```
public class GridService
extends Object
```

This service manages the grid displayed in the `DrawingArea`.

Constructor Summary	Page
GridService(int griddist, int inset) Constructs a new grid with a given griddist and insets.	163
GridService(int width, int height, int griddist, int inset) Constructs a new grid with given width, height, griddist and insets.	162

Method Summary		Page
int	**getGridDist**() Getter for griddist.	165
Point	**getHalfHeightPoint**(Point start, Point end) Returns the midpoint of a vertical line, defined by two points.	164
Point	**getHalfWidthPoint**(Point start, Point end) Returns the midpoint of a horizontal line, defined by two points.	164
int	**getInset**() Getter for insets.	165
Point	**getNextPoint**(DockingPoint dock) Returns a point next to a `DockingPoint`, at that a `GateConnection` should start or end.	164
Point	**getNextPoint**(int x, int y) Returns the point that is next to the specified coordinates.	163
Point	**getNextPoint**(Point p) Returns the point that is next to the specified coordinates.	163
Point	**getNextPointFixX**(double x, double y) Returns the point next to specified coordinates.	164
Point	**getNextPointFixY**(double x, double y) Returns the point next to specified coordinates.	164
Point	**getPoint**(int i, int j) Returns the point at the array position specified by `i` and `j`.	163
Point[][]	**getPoints**() Returns the points of the grid.	163
void	**resize**(int width, int height) Resizes the grid to the specified `width` and `height`.	165

Constructor Detail

GridService

```
public GridService(int width,
                   int height,
                   int griddist,
                   int inset)
```

Constructs a new grid with given width, height, griddist and insets.

Parameters:

`width` - The width of the grid.

`height` - The height of the grid.

`griddist` - The distance between two points of the grid.

`inset` - The insets of the grid.

GridService

```
public GridService(int griddist,
                   int inset)
```

Constructs a new grid with a given griddist and insets. Width and height are 0.

Parameters:

`griddist` - The distance between two points of the grid.

`inset` - The insets of the grid.

Method Detail

getPoints

```
public Point[][] getPoints()
```

Returns the points of the grid.

Returns:

Points of the grid.

---

getPoint

```
public Point getPoint(int i,
                      int j)
```

Returns the point at the array position specified by `i` and `j`.

Parameters:

`i` - The position in the `points` array in the first dimension.

`j` - The position in the `points` array in the second dimension.

Returns:

The point at the specified array position. Or null if `i` or `j` are out of bound.

---

getNextPoint

```
public Point getNextPoint(Point p)
```

Returns the point that is next to the specified coordinates. The coordinates cannot be out of bound.

Parameters:

`p` - A point.

Returns:

The point next to the given point.

---

getNextPoint

```
public Point getNextPoint(int x,
                          int y)
```

Returns the point that is next to the specified coordinates. The coordinates cannot be out of bound.

Parameters:

`x` - The x coordinate in the grid.

`y` - The y coordinate in the grid.

Returns:

The point next to the given coordinates.

---

getNextPoint

```
public Point getNextPoint(DockingPoint dock)
```

Returns a point next to a `DockingPoint`, at that a `GateConnection` should start or end.

Parameters:

`dock` - The `DockingPoint` where a `GateConnection` should start or end.

Returns:

A point in the grid.

---

getNextPointFixY

```
public Point getNextPointFixY(double x,
                              double y)
```

Returns the point next to specified coordinates. The `y` coordinate is taken as fixed, only the `x` coordinate is varied.

Parameters:

`x` - The x coordinate to look for a point.

`y` - The y coordinate to look for a point.

Returns:

The point next to the specified coordinates.

---

getNextPointFixX

```
public Point getNextPointFixX(double x,
                              double y)
```

Returns the point next to specified coordinates. The `x` coordinate is taken as fixed, only the `y` coordinate is varied.

Parameters:

`x` - The x coordinate to look for a point.

`y` - The y coordinate to look for a point.

Returns:

The point next to the specified coordinates.

---

getHalfWidthPoint

```
public Point getHalfWidthPoint(Point start,
                               Point end)
```

Returns the midpoint of a horizontal line, defined by two points.

Parameters:

`start` - The first point.

`end` - The second point.

Returns:

The midpoint between start and end points.

---

getHalfHeightPoint

```
public Point getHalfHeightPoint(Point start,
                                Point end)
```

Returns the midpoint of a vertical line, defined by two points.

Parameters:

`start` - The first point.

`end` - The second point.

Returns:

The midpoint between start and end points.

---

resize

```
public void resize(int width,
                   int height)
```

Resizes the grid to the specified `width` and `height`.

Parameters:

`width` - New width of the grid.

`height` - New height of the grid.

---

getGridDist

```
public int getGridDist()
```

Getter for griddist.

Returns:

The distance between two points of the grid.

---

getInset

```
public int getInset()
```

Getter for insets.

Returns:

The insets of the grid.

service
ImageService

```
java.lang.Object
  └ service.ImageService
```

```
public class ImageService
extends Object
```

Manages all images used by the program. All images can be accessed as public class members.

Field Summary		Page
static Image	**APPLICATION_ICON** Icon of this application.	167
static ImageIcon	**GATE_ICON_AND** Icon of AND gate.	168
static ImageIcon	**GATE_ICON_NOT** Icon of NOT gate.	168
static ImageIcon	**GATE_ICON_OR** Icon of OR gate.	168
static ImageIcon	**GATE_ICON_XOR** Icon of XOR gate.	168
static ImageIcon	**ICON_AND** Icon for AND gate in the `ButtonArea`.	168
static ImageIcon	**ICON_AND_SELECTED** Selected icon for AND gate in the `ButtonArea`.	168
static ImageIcon	**ICON_ARROW_DOWN** Arrow down icon.	168
static ImageIcon	**ICON_ARROW_UP** Arrow up icon.	168
static ImageIcon	**ICON_CHECK_FUNCTION** Icon for checking a Boolean function for satisfiability.	168
static ImageIcon	**ICON_CHECK_FUNCTION_ROLLOVER** Rollover icon for checking a Boolean function for satisfiability.	168
static ImageIcon	**ICON_CHECK_GRAPH** Icon for checking the logic circuit for satisfiability.	169
static ImageIcon	**ICON_CHECK_GRAPH_ROLLOVER** Rollover icon for checking the logic circuit for satisfiability.	169
static ImageIcon	**ICON_CLEAR** Icon for deleting the logic circuit in the `ButtonArea`.	169
static ImageIcon	**ICON_CLEAR_FUNCTION** Icon for clearing a field.	169
static ImageIcon	**ICON_CLEAR_SELECTED** Selected icon for deleting the logic circuit in the `ButtonArea`.	169
static ImageIcon	**ICON_COMPARE** Icon for comparing a Boolean function to a logic circuit.	169
static ImageIcon	**ICON_COMPARE_ROLLOVER** Rollover icon for comparing a Boolean function to a logic circuit.	169
static ImageIcon	**ICON_DELLINE** Icon for deleting a connection in the `ButtonArea`.	169
static ImageIcon	**ICON_DELLINE_SELECTED** Selected icon for deleting a connection in the `ButtonArea`.	169
static ImageIcon	**ICON_EXIT** Icon for exiting the program.	169
static ImageIcon	**ICON_EXPORT** Icon for exporting the logic circuit as an image.	170

static ImageIcon	**ICON_INFO** Icon for displaying program information.	170
static ImageIcon	**ICON_LINE** Icon for creating a connection in the ButtonArea.	170
static ImageIcon	**ICON_LINE_SELECTED** Selected icon for creating a connection in the ButtonArea.	170
static ImageIcon	**ICON_LOAD** Icon for loading a program state.	170
static ImageIcon	**ICON_NOT** Icon for NOT gate in the ButtonArea.	170
static ImageIcon	**ICON_NOT_SELECTED** Selected icon for NOT in the ButtonArea.	170
static ImageIcon	**ICON_OPERAND** Icon for operands in the ButtonArea.	170
static ImageIcon	**ICON_OPERAND_SELECTED** Selected icon for operands in the ButtonArea.	170
static ImageIcon	**ICON_OR** Icon for OR gate in the ButtonArea.	170
static ImageIcon	**ICON_OR_SELECTED** Selected icon for OR gate in the ButtonArea.	171
static ImageIcon	**ICON_RANDOM** Icon for creating a random Boolean function or logic circuit.	171
static ImageIcon	**ICON_RANDOM_SMALL** Icon for creating random truth values.	171
static ImageIcon	**ICON_RANDOM_SMALL_ROLLOVER** Rollover icon for creating random truth values.	171
static ImageIcon	**ICON_REMOVE** Icon for removing a gate in the ButtonArea.	171
static ImageIcon	**ICON_REMOVE_SELECTED** Selected icon for removing a gate in the ButtonArea.	171
static ImageIcon	**ICON_SAVE** Icon for saving the current program state.	171
static ImageIcon	**ICON_XOR** Icon for XOR gate in the ButtonArea.	171
static ImageIcon	**ICON_XOR_SELECTED** Selected icon for XOR in the ButtonArea.	171
static Image	**IMAGE_DOT** Icon of a grid point.	171
static Image	**IMAGE_NEGATION** Icon of a negated input or output of a gate.	172
static Image	**IMAGE_NODE** Icon of a Node.	172
static Image	**OPERAND_ICON** Icon of operand gate.	172

Field Detail

APPLICATION_ICON

```
public static final Image APPLICATION_ICON
```

Icon of this application.

GATE_ICON_AND

`public static final ImageIcon` **`GATE_ICON_AND`**

Icon of AND gate.

GATE_ICON_NOT

`public static final ImageIcon` **`GATE_ICON_NOT`**

Icon of NOT gate.

GATE_ICON_OR

`public static final ImageIcon` **`GATE_ICON_OR`**

Icon of OR gate.

GATE_ICON_XOR

`public static final ImageIcon` **`GATE_ICON_XOR`**

Icon of XOR gate.

ICON_AND

`public static final ImageIcon` **`ICON_AND`**

Icon for AND gate in the `ButtonArea`.

ICON_AND_SELECTED

`public static final ImageIcon` **`ICON_AND_SELECTED`**

Selected icon for AND gate in the `ButtonArea`.

ICON_ARROW_DOWN

`public static final ImageIcon` **`ICON_ARROW_DOWN`**

Arrow down icon.

ICON_ARROW_UP

`public static final ImageIcon` **`ICON_ARROW_UP`**

Arrow up icon.

ICON_CHECK_FUNCTION

`public static final ImageIcon` **`ICON_CHECK_FUNCTION`**

Icon for checking a Boolean function for satisfiability.

ICON_CHECK_FUNCTION_ROLLOVER

`public static final ImageIcon` **`ICON_CHECK_FUNCTION_ROLLOVER`**

Rollover icon for checking a Boolean function for satisfiability.

ICON_CHECK_GRAPH

`public static final ImageIcon` **`ICON_CHECK_GRAPH`**

Icon for checking the logic circuit for satisfiability.

ICON_CHECK_GRAPH_ROLLOVER

`public static final ImageIcon` **`ICON_CHECK_GRAPH_ROLLOVER`**

Rollover icon for checking the logic circuit for satisfiability.

ICON_CLEAR

`public static final ImageIcon` **`ICON_CLEAR`**

Icon for deleting the logic circuit in the `ButtonArea`.

ICON_CLEAR_FUNCTION

`public static final ImageIcon` **`ICON_CLEAR_FUNCTION`**

Icon for clearing a field.

ICON_CLEAR_SELECTED

`public static final ImageIcon` **`ICON_CLEAR_SELECTED`**

Selected icon for deleting the logic circuit in the `ButtonArea`.

ICON_COMPARE

`public static final ImageIcon` **`ICON_COMPARE`**

Icon for comparing a Boolean function to a logic circuit.

ICON_COMPARE_ROLLOVER

`public static final ImageIcon` **`ICON_COMPARE_ROLLOVER`**

Rollover icon for comparing a Boolean function to a logic circuit.

ICON_DELLINE

`public static final ImageIcon` **`ICON_DELLINE`**

Icon for deleting a connection in the `ButtonArea`.

ICON_DELLINE_SELECTED

`public static final ImageIcon` **`ICON_DELLINE_SELECTED`**

Selected icon for deleting a connection in the `ButtonArea`.

ICON_EXIT

`public static final ImageIcon` **`ICON_EXIT`**

Icon for exiting the program.

ICON_EXPORT

`public static final ImageIcon` **`ICON_EXPORT`**

Icon for exporting the logic circuit as an image.

ICON_INFO

`public static final ImageIcon` **`ICON_INFO`**

Icon for displaying program information.

ICON_LINE

`public static final ImageIcon` **`ICON_LINE`**

Icon for creating a connection in the `ButtonArea`.

ICON_LINE_SELECTED

`public static final ImageIcon` **`ICON_LINE_SELECTED`**

Selected icon for creating a connection in the `ButtonArea`.

ICON_LOAD

`public static final ImageIcon` **`ICON_LOAD`**

Icon for loading a program state.

ICON_NOT

`public static final ImageIcon` **`ICON_NOT`**

Icon for NOT gate in the `ButtonArea`.

ICON_NOT_SELECTED

`public static final ImageIcon` **`ICON_NOT_SELECTED`**

Selected icon for NOT in the `ButtonArea`.

ICON_OPERAND

`public static final ImageIcon` **`ICON_OPERAND`**

Icon for operands in the `ButtonArea`.

ICON_OPERAND_SELECTED

`public static final ImageIcon` **`ICON_OPERAND_SELECTED`**

Selected icon for operands in the `ButtonArea`.

ICON_OR

`public static final ImageIcon` **`ICON_OR`**

Icon for OR gate in the `ButtonArea`.

ICON_OR_SELECTED

`public static final ImageIcon` **`ICON_OR_SELECTED`**

Selected icon for OR gate in the `ButtonArea`.

ICON_RANDOM

`public static final ImageIcon` **`ICON_RANDOM`**

Icon for creating a random Boolean function or logic circuit.

ICON_RANDOM_SMALL

`public static final ImageIcon` **`ICON_RANDOM_SMALL`**

Icon for creating random truth values.

ICON_RANDOM_SMALL_ROLLOVER

`public static final ImageIcon` **`ICON_RANDOM_SMALL_ROLLOVER`**

Rollover icon for creating random truth values.

ICON_REMOVE

`public static final ImageIcon` **`ICON_REMOVE`**

Icon for removing a gate in the `ButtonArea`.

ICON_REMOVE_SELECTED

`public static final ImageIcon` **`ICON_REMOVE_SELECTED`**

Selected icon for removing a gate in the `ButtonArea`.

ICON_SAVE

`public static final ImageIcon` **`ICON_SAVE`**

Icon for saving the current program state.

ICON_XOR

`public static final ImageIcon` **`ICON_XOR`**

Icon for XOR gate in the `ButtonArea`.

ICON_XOR_SELECTED

`public static final ImageIcon` **`ICON_XOR_SELECTED`**

Selected icon for XOR in the `ButtonArea`.

IMAGE_DOT

`public static final Image` **`IMAGE_DOT`**

Icon of a grid point.

IMAGE_NEGATION

```
public static final Image IMAGE_NEGATION
```

Icon of a negated input or output of a gate.

IMAGE_NODE

```
public static Image IMAGE_NODE
```

Icon of a `Node`.

OPERAND_ICON

```
public static final Image OPERAND_ICON
```

Icon of operand gate.

service
LogicService

```
java.lang.Object
  └ service.LogicService
```

public class **LogicService**
extends Object

Service managing all boolean operations in the program.

Constructor Summary		Page
	LogicService()	173
Method Summary		**Page**
boolean	**compare**(String function, DrawableComponent rootGate) Compares a given boolean function and a Graph.	174
boolean	**compare**(String function1, String function2) Compares two Boolean functions.	174
boolean	**evaluateFunction**(String function) Evaluates a boolean Function.	174
boolean	**evaluateGates**(DrawableComponent root) Evaluates the Graph thats root DrawableComponent is specified.	174
BoolTree	**functionToTree**(String function) Builds a BoolTree out of a boolean function.	173
String	**getGraphFunction**() Returns graphTree as a function string.	174
BoolTree	**graphToTree**(DrawableComponent root) Builds a BoolTree out of a Graph.	174
boolean	**isSatisfiable**(Graph graph) Checks if a specified Graph is satisfiable.	175
boolean	**isSatisfiable**(String function) Checks if a specified Boolean function is satisfiable.	175
void	**randomValues**() Creates random truth values for all operands.	175

Constructor Detail

LogicService

public **LogicService**()

Method Detail

functionToTree

public BoolTree **functionToTree**(String function)

Builds a BoolTree out of a boolean function. Construction of the tree and analysis of function is done in BoolTree.

Parameters:

function - Function to analyze.

Returns:

The BoolTree representing function.

graphToTree

`public BoolTree graphToTree(DrawableComponent root)`

Builds a `BoolTree` out of a `Graph`. Uses `graphToTreeRecursive` to build the tree.

Parameters:

`root` - The root `DrawableComponent` of the `Graph` to analyze.

Returns:

The `BoolTree` representing a `Graph`.

---

compare

`public boolean compare(String function, DrawableComponent rootGate)`

Compares a given boolean function and a `Graph`.

Parameters:

`function` - Boolean function to compare.

`rootGate` - Root `DrawableComponent` of the `Graph` that is to be compared.

Returns:

`True` if `function` and `Graph` are equal.

---

getGraphFunction

`public String getGraphFunction()`

Returns `graphTree` as a function string.

Returns:

`graphTree` as a function or an empty String if `graphTree` is null.

---

evaluateFunction

`public boolean evaluateFunction(String function)`

Evaluates a boolean Function. Logical values of the `Qualifyer`s are taken for evaluation. Evaluation is done in `BoolTree.evaluate()`.

Parameters:

`function` - Function to evaluate.

Returns:

`True` if the function evaluates to `true`. `False` otherwise.

---

evaluateGates

`public boolean evaluateGates(DrawableComponent root)`

Evaluates the `Graph` thats root `DrawableComponent` is specified. Logical values of the `Qualifyer`s are taken for evaluation. Evaluation is done in `BoolTree.evaluate()`.

Parameters:

`root` - The root `DrawableComponent` of the `Graph` that is to be evaluated.

Returns:

`True` if the `Graph` evaluates to `true`. `False` otherwise.

---

compare

```
public boolean compare(String function1,
                       String function2)
```

Compares two Boolean functions. Comparison is done in `BoolTree.equals()`.

Parameters:

`function1` - First function to compare.

`function2` - Second function to compare.

Returns:

`True` if `function1` is equal to `function2`.

---

randomValues

```
public void randomValues()
```

Creates random truth values for all operands.

---

isSatisfiable

```
public boolean isSatisfiable(String function)
```

Checks if a specified Boolean function is satisfiable.

Parameters:

`function` - Function to check.

Returns:

`True` if the function is satisfiable. `False` otherwise.

---

isSatisfiable

```
public boolean isSatisfiable(Graph graph)
```

Checks if a specified `Graph` is satisfiable.

Parameters:

`graph` - `Graph` to check.

Returns:

`True` if the `Graph` is satisfiable. `False` otherwise.

service
QualifyerService

```
java.lang.Object
  └ service.QualifyerService
```

```
public class QualifyerService
extends Object
```

Service that manages `Qualifyer`s for the program. All `Qualifyer`s can be accessed as public class members. `QualifyerService` cannot be instantiated.

Field Summary		Page
static QualifyerGate	**Q_AND** Qualifyer representing AND gate.	176
static ButtonQualifyer	**Q_DELLINE** Qualifyer representing DELETE LINE action.	176
static ButtonQualifyer	**Q_LINE** Qualifyer representing CREATE LINE action.	177
static QualifyerGate	**Q_NAND** Qualifyer representing NAND gate.	177
static QualifyerGate	**Q_NOR** Qualifyer representing NOR gate.	177
static QualifyerGate	**Q_NOT** Qualifyer representing NOT gate.	177
static QualifyerGate	**Q_OR** Qualifyer representing OR gate.	177
static ButtonQualifyer	**Q_REMOVE** Qualifyer representing REMOVE action.	177
static QualifyerGate	**Q_XOR** Qualifyer representing XOR gate.	177

Method Summary		Page
static BooleanQualifyer	**exchangeQualifyer**(BooleanQualifyer qualifyer) Exchanges a `Qualifyer` for an equal `Qualify`.	178
static ArrayList<QualifyerOperand>	**getOperandQualifyers**() Getter for `operands`.	177
static ArrayList<QualifyerGate>	**getOperatorQualifyers**() Returns all `Qualifyer`s that represent a logical operator apart from NOT.	178
static BooleanQualifyer	**getQualifyerByChar**(char c) Returns the `Qualifyer` representing a character that represents an operand or operator.	177

Field Detail

Q_AND

```
public static final QualifyerGate Q_AND
```

Qualifyer representing AND gate.

Q_DELLINE

```
public static final ButtonQualifyer Q_DELLINE
```

Qualifyer representing DELETE LINE action.

Q_LINE

`public static final ButtonQualifyer` **`Q_LINE`**

Qualifyer representing CREATE LINE action.

Q_NAND

`public static final QualifyerGate` **`Q_NAND`**

Qualifyer representing NAND gate.

Q_NOR

`public static final QualifyerGate` **`Q_NOR`**

Qualifyer representing NOR gate.

Q_NOT

`public static final QualifyerGate` **`Q_NOT`**

Qualifyer representing NOT gate.

Q_OR

`public static final QualifyerGate` **`Q_OR`**

Qualifyer representing OR gate.

Q_REMOVE

`public static final ButtonQualifyer` **`Q_REMOVE`**

Qualifyer representing REMOVE action.

Q_XOR

`public static final QualifyerGate` **`Q_XOR`**

Qualifyer representing XOR gate.

Method Detail

getQualifyerByChar

`public static BooleanQualifyer` **`getQualifyerByChar`**`(char c)`

Returns the `Qualifyer` representing a character that represents an operand or operator. If no such `Qualifyer` exists and the character is a letter, a new `QualifyerOperand` is created.

Parameters:

`c` - character of an operator or operand.

Returns:

`Qualifyer` that represents the char.

---

getOperandQualifyers

`public static ArrayList<QualifyerOperand> getOperandQualifyers()`

Getter for `operands`. Collection is initialized before return if not already is.

Returns:

Collection of all `QualifyerOperand`s used in this program.

---

getOperatorQualifyers

`public static ArrayList<QualifyerGate> getOperatorQualifyers()`

Returns all `Qualifyer`s that represent a logical operator apart from NOT.

Returns:

All `Qualifyer`s that represent a logical operator apart from NOT.

---

exchangeQualifyer

`public static BooleanQualifyer exchangeQualifyer(BooleanQualifyer qualifyer)`

Exchanges a `Qualifyer` for an equal `Qualify`. This is necessary when a `Qualifyer` is loaded from a file.

Parameters:

`qualifyer` - `Qualifyer` to exchange.

Returns:

A `Qualifyer` equal to the passed one.

Package

utils

Class Summary		Page
BoolTree	A BoolTree saves the information about a Boolean function.	179
SerializationContainer	Transfer object used for serialization of the current state of the program.	184

utils
BoolTree

```
java.lang.Object
  └ utils.BoolTree
```

```
public class BoolTree
extends Object
```

A BoolTree saves the information about a Boolean function. The information is saved in a tree structure and a couple of logical operations such as comparison with another Boolean function, represented by a BoolTree and construction of a new BoolTree out of a Boolean function represented by a String are provided.

Nested Class Summary		Page
abstract class	BoolTree.Element Element of the tree.	181

Field Summary		Page
BoolTree.Element	root The root element of this tree.	180

Constructor Summary	Page
BoolTree()	180

Method Summary		Page
boolean	analyze(String function) Analyzes a Boolean function.	180
BoolTree.Element	createNewElement(BooleanQualifyer q) Creates a new element of the tree depending on the Qualifyer passed.	181
BoolTree.Element	createNewElement(BooleanQualifyer q, boolean negated) Creates a new element of the tree depending on the Qualifyer passed.	181
boolean	equals(Object obj) Checks if two BoolTrees represent the same Boolean function.	181
boolean	evaluate() Evaluates a BoolTree.	180
HashSet<QualifyerOperand>	getOperands() Returns all operands that occur in this BoolTree.	180
boolean	isSatisfiable() Checks if the Boolean function that is represented by this BoolTree is satisfiable.	181
String	toString() Returns the Boolean function that is represented by this BoolTree.	180

Field Detail

root

`public BoolTree.Element root`

The root element of this tree.

Constructor Detail

BoolTree

`public BoolTree()`

Method Detail

evaluate

`public boolean evaluate()`

Evaluates a `BoolTree`. Evaluation is done in each `Element` of the tree. For evaluation, the logical truths of each operand is taken from its corresponding `QualifyerOperand`.

Returns:

`True` if this trees current logic truth is `true`. `False` otherwise.

---

analyze

`public boolean analyze(String function)`

Analyzes a Boolean function. If the given String represents a valid function, a corresponding tree is built from it. Building of the tree is done in `buildTree(String function)`.

Parameters:

`function` - The Boolean function to analyze.

Returns:

`True` if the function was analyzed. `False` otherwise.

---

toString

`public String toString()`

Returns the Boolean function that is represented by this `BoolTree`. If no tree has been built yet, `null` is returned.

Overrides:

`toString` in class `Object`

---

getOperands

`public HashSet<QualifyerOperand> getOperands()`

Returns all operands that occur in this `BoolTree`.

Returns:

All operands that occur in this `BoolTree`.

---

equals

`public boolean` **`equals`**`(Object obj)`

Checks if two `BoolTree`s represent the same Boolean function.

Overrides:

`equals` in class `Object`

Parameters:

`obj` - An objects that this `BoolTree` is compared to.

Returns:

`True` if `obj` is a `BoolTree` and represents the same Boolean function as this `BoolTree` does. `False` otherwise.

---

isSatisfiable

`public boolean` **`isSatisfiable`**`()`

Checks if the Boolean function that is represented by this `BoolTree` is satisfiable.

Returns:

`True` if the function is satisfiable. `False` otherwise.

---

createNewElement

`public BoolTree.Element` **`createNewElement`**`(BooleanQualifyer q)`

Creates a new element of the tree depending on the `Qualifyer` passed.

- A `QualifyerOperator` will return a new Operator.
- If the `Qualifyer` represents the NOT sign, a new NOT is returned.
- A `QualifyerOperand` will return a new Operand.

Parameters:

`q` - `Qualifyer` representing the same part of a Boolean function that the returned element represents.

Returns:

A new element of the `BoolTree` representing the same part of a Boolean function `Qualifyer q` does.

---

createNewElement

`public BoolTree.Element` **`createNewElement`**`(BooleanQualifyer q, boolean negated)`

Creates a new element of the tree depending on the `Qualifyer` passed.

- A `QualifyerOperator` will return a new Operator.
- If the `Qualifyer` represents the NOT sign, a new NOT is returned.
- A `QualifyerOperand` will return a new Operand.

Parameters:

`q` - `Qualifyer` representing the same part of a Boolean function that the returned element represents.

`negated` - If `negated` is `True`, the output of the element will be negated.

Returns:

A new element of the `BoolTree` representing the same part of a Boolean function `Qualifyer q` does.

utils
BoolTree.Element

```
java.lang.Object
  └ utils.BoolTree.Element
```

Enclosing class:

BoolTree

```
public abstract class BoolTree.Element
extends Object
```

Element of the tree. Each element represents either an operator or an operand.

Field Summary		Page
BoolTree.Element	**left** The Element representing the left input of this Element.	182
BoolTree.Element	**right** The Element representing the right input of this Element.	182

Constructor Summary	Page
BoolTree.Element(BooleanQualifyer q) Constructs a new Element with the given Qualifyer.	183

Method Summary		Page
abstract boolean	**evaluate**() Evaluates this Element.	183
BooleanQualifyer	**getQualifyer**() Returns the Qualifyer of this element.	184
abstract boolean	**isLeaf**() Returns True if this Element represents an operand.	183
boolean	**isNegated**() Returns True if the output of this element is negated or False otherwise.	184
boolean	**isNegatedLeft**() Returns True if the left input of this element is negated or False otherwise.	184
boolean	**isNegatedRight**() Returns True if the right input of this element is negated or False otherwise.	184
void	**setNegated**(boolean neg) Sets the negation of the output to neg.	183
void	**setNegateLeft**(boolean neg) Sets the negation of the left input to neg.	183
void	**setNegateRight**(boolean neg) Sets the negation of the right input to neg.	183
String	**toString**() Returns a String containing the Boolean function this element and its children represent.	184

Field Detail

left

```
public BoolTree.Element left
```

The Element representing the left input of this Element.

---

right

```
public BoolTree.Element right
```

The `Element` representing the right input of this `Element`.

Constructor Detail

BoolTree.Element

`public` **`BoolTree.Element`**`(`BooleanQualifyer `q)`

Constructs a new Element with the given `Qualifyer`.

Parameters:

`q` - The `Qualifyer` specifying what operator or operand this `Element` represents.

Method Detail

setNegated

`public void` **`setNegated`**`(boolean neg)`

Sets the negation of the output to `neg`.

Parameters:

`neg` - `True` if the output should be negated. `False` otherwise.

---

setNegateLeft

`public void` **`setNegateLeft`**`(boolean neg)`

Sets the negation of the left input to `neg`.

Parameters:

`neg` - `True` if the left input should be negated. `False` otherwise.

---

setNegateRight

`public void` **`setNegateRight`**`(boolean neg)`

Sets the negation of the right input to `neg`.

Parameters:

`neg` - `True` if the right input should be negated. `False` otherwise.

---

evaluate

`public abstract boolean` **`evaluate`**`()`

Evaluates this `Element`.

Returns:

`True` if the logic value of this `Element` is true. `False` otherwise.

---

isLeaf

`public abstract boolean` **`isLeaf`**`()`

Returns `True if this Element represents an operand. False if not.`

Returns:

`True if this Element represents an operand. False if not.`

---

toString

`public String` **`toString`**`()`

Returns a String containing the Boolean function this element and its children represent.

Overrides:

`toString` in class `Object`

---

getQualifyer

`public` `BooleanQualifyer` **`getQualifyer`**`()`

Returns the `Qualifyer` of this element.

Returns:

The `Qualifyer` of this element.

---

isNegatedRight

`public boolean` **`isNegatedRight`**`()`

Returns `True` if the right input of this element is negated or `False` otherwise.

Returns:

`True` if the right input of this element is negated or `False` otherwise.

---

isNegatedLeft

`public boolean` **`isNegatedLeft`**`()`

Returns `True` if the left input of this element is negated or `False` otherwise.

Returns:

`True` if the left input of this element is negated or `False` otherwise.

---

isNegated

`public boolean` **`isNegated`**`()`

Returns `True` if the output of this element is negated or `False` otherwise.

Returns:

`True` if the output of this element is negated or `False` otherwise.

utils
SerializationContainer

```
java.lang.Object
  └ utils.SerializationContainer
```

All Implemented Interfaces:

Serializable

```
public class SerializationContainer
extends Object
implements Serializable
```

Transfer object used for serialization of the current state of the program. This is a `Graph` object, a Boolean function entered by the user and a Boolean function representing the `Graph`.

Constructor Summary	Page
SerializationContainer()	185

Method Summary		Page
String	**getFunction**() Returns the user-entered function.	186
String	**getGateFunction**() Returns the Boolean function equaling the `Graph`.	186
Graph	**getGraph**() Getter for the `Graph`.	185
void	**setFunction**(String function) Sets the user-entered function.	186
void	**setGateFunction**(String gateFunction) Set the Boolean function equaling the `Graph`.	186
void	**setGraph**(Graph graph) Setter for the `Graph`.	185

Constructor Detail

SerializationContainer

```
public SerializationContainer()
```

Method Detail

getGraph

```
public Graph getGraph()
```

Getter for the `Graph`.

Returns:

The `Graph` of this container.

---

setGraph

```
public void setGraph(Graph graph)
```

Setter for the `Graph`.

Parameters:

`graph` - The new `Graph` of this container.

---

getFunction

`public String` **`getFunction`**`()`

Returns the user-entered function.

Returns:

The user-entered function.

---

setFunction

`public void` **`setFunction`**`(String function)`

Sets the user-entered function.

Parameters:

`function` - The new user-entered function.

---

getGateFunction

`public String` **`getGateFunction`**`()`

Returns the Boolean function equaling the `Graph`.

Returns:

The Boolean function equaling the `Graph`.

---

setGateFunction

`public void` **`setGateFunction`**`(String gateFunction)`

Set the Boolean function equaling the `Graph`.

Parameters:

`gateFunction` - The new Boolean function equaling the `Graph`.

Lebenslauf

Persönliche Daten:

Name	Markus Bauer
Geburtsdatum	14.5.1987
Staatsangehörigkeit	Österreich

Ausbildung:

1993 bis 1997	Volksschule, Wien
1997 bis 2005	Bundesrealgymnasium Wien XIX Krottenbachstraße
2005	Reifeprüfung
2005 bis 2006	Zivildienst
2006 bis 2012	Universität Wien, Unterrichtsfach Physik und Informatik